TEKNILLINEN KORKEAKOULU
Sähkö- ja tietoliikennetekniikan osasto

Mika Laurell

Palvelun toteuttaminen hajautetussa palvelualustassa

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 19.8.2002.

Valvoja	Professori	Seppo J. Halme
Ohjaaja	DI	Pasi Juhava

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä:	Mika Laurell
Työn nimi:	Palvelun toteuttaminen hajautetussa palvelualustassa
Päivämäärä:	19.8.2002 **Sivumäärä**: 142
Osasto:	Sähkö- ja tietoliikennetekniikan osasto
Professuuri:	S-72 Tietoliikennetekniikka
Työn valvoja:	Professori Seppo J. Halme, Teknilleinen Korkeakoulu
Työn ohjaaja:	DI Pasi Juhava, Elisa Communications Oyj

Työssä tutustutaan OSGi (Open Service Gateway Iniative) -palvelualustamääritelmään sekä luodaan palveluesimerkki käyttäen OSGi-viitekehystä. Tämä työ toimii pohjana mietittäessä miten operaattori voisi toteuttaa palveluita OSGi-palvelualustaympäristössä.

OSGi-palvelualusta tarjoaa palvelun suunnittelijalle monia mahdollisuuksia toteuttaa haluttu palvelu. Toisaalta se kuitenkin määrittelee joukon rajapintoja, jotka suunnittelijan on toteutettava. Jotta voisimme toteuttaa palveluesimerkin OSGi-määritysten mukaan, on palvelualustan olennaisimpien kohtien ymmärtäminen välttämätöntä. Tämän vuoksi työssä on annettu varsin suuri painoarvo juuri OSGi-palvelualustan toiminnan ja vaatimusten kuvaamiselle.

Palvelut OSGi-palvelualustaan kirjoitetaan käyttäen Java-ohjelmointikieltä. Palveluesimerkki on tehty vastaamaan oikeaa palvelua, joka tässä työssä on kodin vartiointipalvelu. Työssä kuvataan myös kuinka useita palvelualustoja sisältävää verkkoa voidaan hallita mielekkäästi sekä esitellään palvelun toimittamiseen läheisesti liittyvien toimijoiden roolit.

Työn tavoitteena on ensinnäkin saada kuva siitä millaisia vaatimuksia palveluiden toimittaminen asiakkaalle asettaa operaattorille. Toisena tavoitteena on selvittää onko OSGi:n määritys jo siinä vaiheessa, että operaattori voi aloittaa palveluiden tarjoamisen asiakkailleen.

Avainsanat: OSGi, Palvelualusta, Bundle, Palvelu, Palvelualustan hallinta, Java

HELSINKI UNIVERSITY OF TECHNOLOGY **ABSTRACT OF THE MASTER'S THESIS**

Author:	Mika Laurell
Name of the thesis:	Creation of services for distributed service platform
Date:	19.8.2002 **Number of pages**: 142
Faculty:	Department of Electrical and Communications Engineering
Professorship:	S-72 Communications
Supervisor:	Professor Seppo J. Halme, Helsinki University of Technology
Instructor:	MSc Pasi Juhava, Elisa Communications Corporation

This study introduces the OSGi (Open Service Gateway Initiative) service platform specification and includes the creation of a proof-of-concept service by utilising one of the commercial OSGi framework platforms. This study will act as the basis for steering further studies of the OSGi technologies and in positioning Elisa Communications in the OSGi service provider field.

The OSGi service platform provides the application designer with multiple implementation possibilities, but at the same time standardises some required interfaces in every application to unify the communication API's.

To be able to create a sample application the researcher has to gain first detailed knowledge of the OSGi platform and specifications. Therefore, a large part of this Master's Thesis is devoted to describing the OSGi service platform and its operational requirements.

All OSGi conformant services are programmed in the Java(tm) programming language. The home security service application implemented in this thesis is taken directly from the real world and has been reimplemented using the OSGi environment. Part of this thesis discusses the problematics of management in an network of distributed service platforms and clarifies the different roles of players in the service delivery network used with OSGi-service gateways.

The goal of this work is to obtain a clear picture of the requirements for acting as an service operator for customers and to find out if the OSGi specification is mature enough to enable fullscale commercial operation in the service provider segment.

Keywords: OSGi, Service Platform, Bundle, Service, Remote Management, Java

ALKULAUSE

Diplomityötä valvoi prof. Seppo J. Halme Teknillisen korkeakoulun tietoliikennelaboratoriosta. Diplomityö on tehty Elisa Communications'in Elisa Innovations-yksikkössä ja sen kirjoittamista ohjasi DI Pasi Juhava. Haluan kiittää molempia heidän ohjauksestaan diplomityön kirjoituksen aikana.

Haluan lisäksi kiittää kaikkia Elisa Research Center'in henkilökunnasta, jotka ovat olleet mukana diplomityön eri vaiheissa antamassa uusia näkökulmia käsiteltäviin asioihin sekä heidän kannustuksestaan.

Parhaat kiitokset perheelleni ja vaimolleni Anulle opiskeluaikanani saamasta kannustuksesta.

Espoossa 19.8.2002

Mika Laurell

SISÄLLYSLUETTELO

Lyhenteet

ADSL	*Asymmetric Digital Subscriber Line, asymmetrinen digitaalinen tilaajaliittymä*
AEG	*Architecture Expert Group, arkkitehtuuriasiantuntijaryhmä*
API	*Aplication Programming Interface, sovellusliittymä*
CDR	*Charge Detail Record, laskentatietue*
CE	*Charge Event, laskentatapahtuma*
CED	*Charge Event Description, laskentatapahtumakuvaus*
COM	*serial COMmunications port, tietokoneen sarjaportti*
CPE	*Customer Premises Equipment, asiakkaan päätelaite*
CPEG	*Core Platform Expert Group, alusta-asiantuntijaryhmä*
DEG	*Device Expert Group, laiteasiantuntijaryhmä*
DHCP	*Dynamic Host Configuration Protocol, dynaaminen IP-osoitteiden jakoprotokolla*
DSL	*Digital Subscriber Line, digitaalinen tilaajajohto*
DVB	*Digital Video Broadcasting, digitaalinen televisio-ohjelmien lähetystekniikka*
EG	*Expert Group, asiantuntijaryhmä*
EIG	*Entertainment Interest Group, viihteen intressiryhmä*
HAVi	*Home Audio and Video Interoperability, kodin laitteiden välinen rajapinta*
HomePNA	*Home Phoneline Networking Alliance, puhelinkaapelointia käyttävä tiedonsiirtotekniikka*
HTML	*HyperText Markup Language, WWW:n käyttämä sivunkuvauskieli*
HTTP	*HyperText Transport Protocol, WWW -palvelun käyttämä protokolla hypermediadokumenttien siirtämiseksi Internet -verkossa.*
IEEE	*The Institute of Electical and Electronics Engineers, sähkötekniikanalan insinöörien kattojärjestö, joka tekee muun muassa standardointi-työtä*
IEEE 1394	*Firewire, nopea liitäntä tietokoneen ja oheislaitteen esim. videokameran tai kovalevyn välillä.*

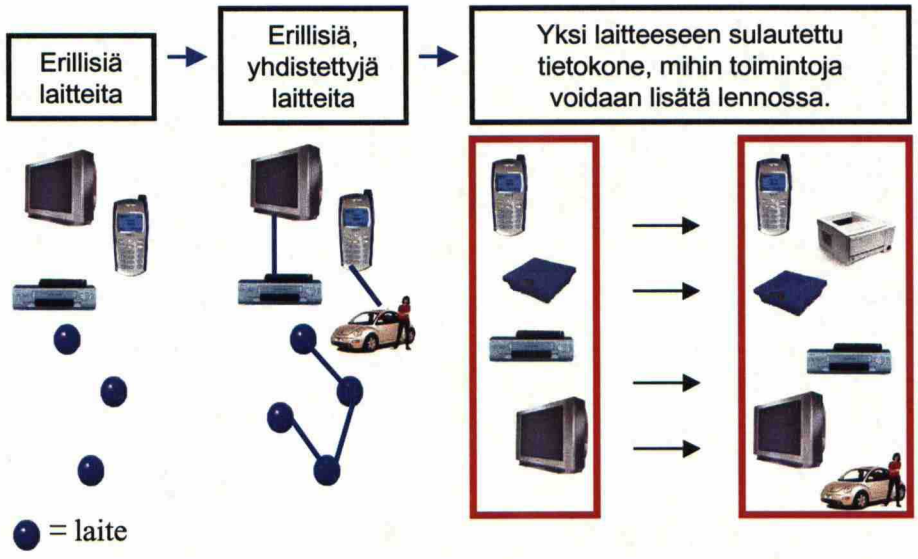
IETF	*Internet Engineering Task Force, IAB'in (Internet Activities Board) alainen vapaaehtoisista henkilöistä koostuva Internet-verkon kehitysryhmä.*
IP	*Internet Protocol, TCP/IP-perheen verkkokerroksen protokolla*
J2SE	*Java 2 platform, Standard Edition, Java alustan versio 2*
JAAS	*Java Java Authentication and Authorization Service, Javan autentikointi- ja auktorisointipalvelu*
JAR	*Java ARchive, Java-arkistotiedosto*
Java	*Sun Microsystemsin kehittämä laitteistoriippumaton oliopohjainen ohjelmointikieli*
JDK	*Java Development Kit, Java kehitysympäristö*
Jini	*Java-pohjainen laitteiden liittämiseen ja niiden väliseen tiedonsiirtoon tarkoitettu tekniikka*
JVM	*Java Virtual Machine, Java-virtuaaliympäristö*
JXTA	*JuXTApose, avoin protokolla vertaisverkkoihin, alun perin Sun Microsystems'in kehittämä*
LAN	*Local Area Network, lähiverkko*
LDAP	*Lightweight Directory Access Protocol, kevyt tietokantahakuprotokolla*
LON	*Local Operating Network, Echelonin kehittämä älykäs ja hajautettu tieto- ja automaatiojärjestelmä.*
MGCP	*Media Gateway Control Protocol, puhelun välityksessä IP-verkossa käytetty protokolla*
MRD	*Marketing Requirement Document, markkinoiden vaatimusdokumentti*
NGN	*Next Generation Networks, seuraavan sukupolven puhelinverkko*
OSG	*Open Service Gateway, avoin palveluportti*
OSGi	*Open Service Gateway Iniative, avoimen palveluporttin määrityksen tehnyt yhteisö*
PDA	*Personal Digital Assistant, kämmentietokone*
PKI	*Public Key Infrastructure, julkisen avaimen järjestelmä*
PS	*PostScript, Adobe Systems'n kehittämä tiedostoformaatti*
RFC	*Request for Comment, IETF:n tai OSGi:n suositus*
RFP	*Request for Proposal, ehdotus IETF:n tai OSGi:n suositukseksi*
RGW	*Residential Gateway, palveluportti*
RMEG	*Remote Management Expert Group, etähallinta-asiantuntijaryhmä*

SEG	*Security Expert Group, turvallisuusasinantuntijaryhmä*
SGW	*Service Gateway, palveluportti*
SSL	*Secure Sockets Layer, WWW:n tietoliikenteen salausmenetelmä.*
TFTP	*Trivial File Transfer Protocol, yksinkertainen tiedoston siirto protokolla*
TSC	*Technical Steering Committee, tekninen ohjausryhmä*
UPnP	*Universal Plug and Play, Microsoft'in kehittelemä laitteiden liittämistekniikka*
USB	*Universal Serial Bus, nopea sarjaväylä*
VEG	*Vechile Expert Group, ajoneuvoasiantintijaryhmä*
WCDMA	*Wideband Code Division Multiple Access, laajakaistainen koodijakoinen monikanavointitekniikka*
WLAN	*Wireles Local Area Network, langaton lähiverkko, perustuu määritykseen IEEE 802.11b*

1. JOHDANTO

1.1 Toimintaympäristö

Internet-yhteyksien kehitys, tietokoneiden hintojen lasku ja kodin älykkäiden laitteiden nopea lisääntyminen ovat merkittävästi kasvattaneet kiinnostusta kotiautomaatioon. Kuluttajaelektroniikkatuotteiden valmistajat lisäävät tuotteisiinsa yhä enemmän älykkyyttä, mikä sallii laitteiden verkottamisen niin, että niitä voidaan hallita keskitetysti (Kuva 1). Lisäksi langattoman tiedonsiirtoteknologian kehitys on tuonut kotiin uusia päätelaitteita, kuten kämmentietokoneita.



Kuva 1. Viime vuosina tapahtunut muutos laitearkkitehtuurissa. [1]

Nykyisin yhä useammassa kodissa on useampi kuin yksi tietokone ja jatkuva laajakaistainen Internet-yhteys. Tämä on luonut uusia vaatimuksia, kuinka laitteita tulisi voida käyttää yhdessä. Esimerkkeinä näistä vaatimuksista ovat: usean kotikäyttäjän yhtäaikainen yhteys Internetiin, tiedostojen ja laitteiden jako käyttäjien kesken, kodin hallinta ja automaatio, kodin ja työpaikan välinen yhteys, kodin valvonta sekä videovirran jako kodin sisällä tai haluttuun paikkaan kodin ulkopuolella. [2]

Edellä esitetyt vaatimukset esittelevät markkinat uudenlaisille tuotteille, joita voidaan kutsua kodin palveluporteiksi (Service Gateway, SGW, Residential

Gateway, RGW). Palveluportti tarjoaa tarvittavat yhteydet, jotta käyttäjä voi käyttää hyväkseen kotinsa verkotettuja laitteita. Tämän lisäksi palveluportti tarjoaa palvelualustan kodin laitteita älykkäästi käyttäville palveluille esimerkiksi tilausvideopalvelu, kodin turvapalvelut, kodin huoltokirja, erilaiset mittarit kuten energian kulutuksen seuranta sekä virtuaaliverkot kodin ja työpaikan välillä.

1.2 Tulevaisuuden palvelumalli

Tänä päivänä useissa eri yhteyksissä puhutaan palveluista. Kaikkea uutta teknologiaa yritetään kaupata sen tarjoamilla uusilla palveluilla. Hyvä esimerkki tästä on digitaalinen televisio (Digital Video Broadcasting, DVB), minkä yhteydessä on paljon puhuttu vuorovaikutteisista palveluista.

Mikä on palvelu? Perinteisessä mielessä palvelu on aina työn tekemistä toisen puolesta. Nykyisin palvelun käsite on hieman laajentunut käsittäen myös tietokoneen tai muun vastaavan suorittaman tehtävän, kuten tekstinkäsittelyn. Usein näissä palveluissa palvelun käyttäjä on aktiivisena osapuolena mukana.

Jotta uusia palveluita käytettäisiin, tulee niiden olla helposti saatavilla. Tulevaisuudessa yhä useammat palvelut voidaan ladata tietokoneeseen tai palveluporttiin vasta, kun niitä tarvitaan ja poistaa, kun niitä ei enää käytetä. Näin ei tuhlata palveluportin mahdollisesti hyvinkin rajallisia resursseja.

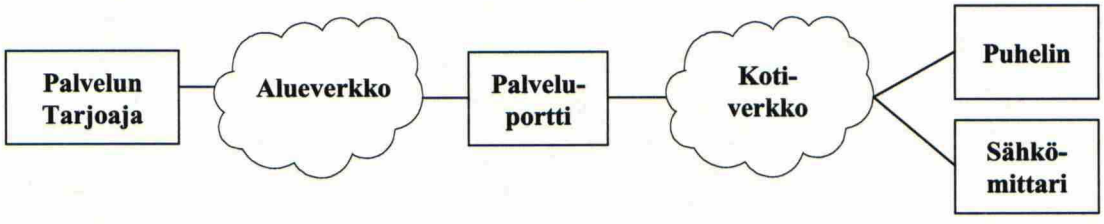
1.3 OSGi (Open Service Gateway Initiative)

Yhden palveluportin toteutusvaihtoehdoista on määritellyt Open Service Gateway Iniative (OSGi) niminen yhteenliittymä, joka perustettiin maaliskuussa 1999. Se on teollisuusryhmä, joka työskentelee määritelläkseen ja edistääkseen avointa standardia tulevien älykkäiden koti- ja pientoimistolaitteiden yhteenliittämiseksi Internetin välityksellä. OSGi:n ajatuksena on määritellä mahdollisimman avoin, mutta kuitenkin standardoitu järjestelmä, mikä toimisi eräänlaisina sovittimena kodin eri laitetekniikoiden välillä. Lisäämällä sovitinrajapintaan älykkyyttä, OSGi mallin mukaisella laitteella pystytään tuottamaan monipuolisia palveluita käyttäen hyväksi kodin olemassa olevaa laitekantaa. OSGi ei määrittele itse laitetta,

palveluporttia, vaan keskittyy palveluportin ohjelmiston, palvelualustan, määrittelemiseen. [3]

OSGi:llä on kolme avainkohtaa heidän suunnitelmissaan; useat yhtäaikaiset palvelut, alueverkot ja kotiverkot sekä laitteet. Toisin kuin muut suositukset (initiatives), OSGi keskittyy päästä päähän ratkaisuarkkitehtuureihin, alkaen palvelun tarjoajasta (service provider) ja päättyen paikallisiin laitteisiin. Koska OSGi-määritys keskittyy tarjoamaan avoimen sovelluskerroksen ja palveluporttirajapinnan, määritys käsittää ja laajentaa käytännössä kaikkia tämän hetkisiä verkkostandardeja ja suosituksia. OSGi-määritys tarjoaa palvelun tarjoajille, järjestelmäsuunnittelijoille, ohjelmistotoimittajille, laitetoimittajille ja laitevalmistajille yhteisen avoimen arkkitehtuurin suunnitella, ottaa käyttöön ja hallita useita palveluita koordinoidusti. [4]

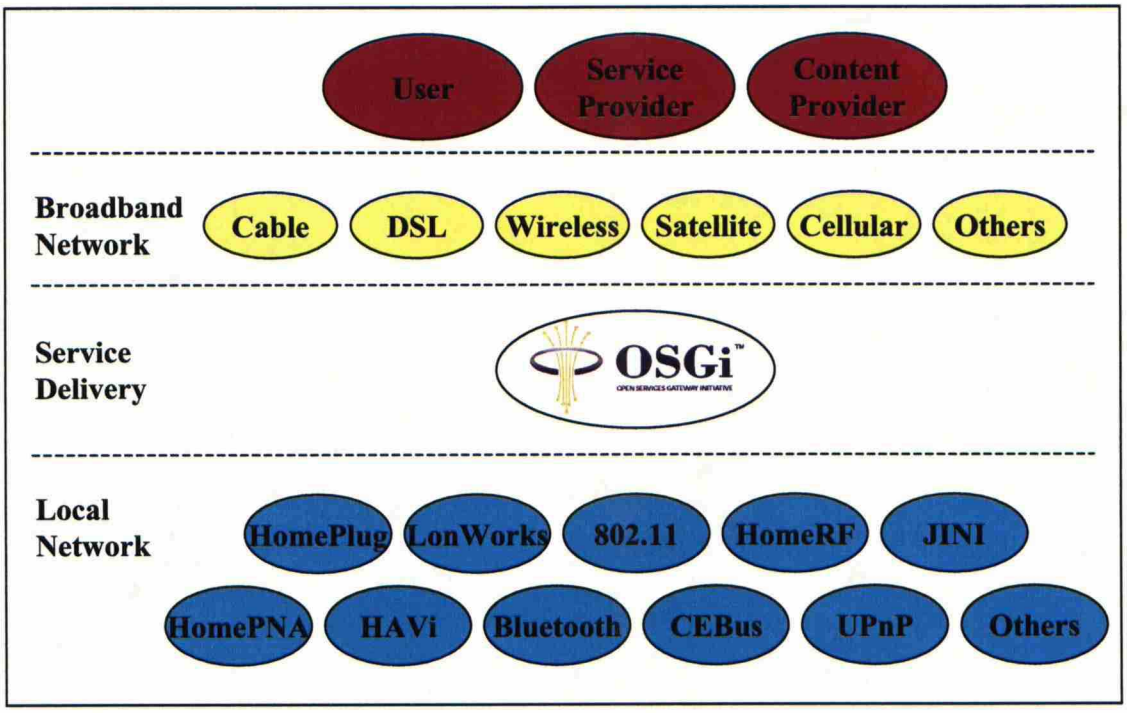
Merkittävimpänä komponenttina OSGi:n määrityksessä on palveluportti (Service Gateway), joka toimii alustana useille tiedonsiirtoon perustuville palveluille. Palveluportti voi ottaa käyttöön, yhdistellä ja hallita sekä tulevia että lähteviä puhe-, data-, Internet- ja multimediayhteyksiä. Palveluportti voi toimia myös sovelluspalvelimena. Esimerkkeinä tällaisista sovelluksista voisivat olla terveydenhuollon tarkkailupalvelu, vartiointiliikkeen valvontapalvelu, kodin laitteiden hallintapalvelu, sähköisen kaupankäynnin palvelut sekä sähköyhtiöiden palvelut, kuten sähkömittarin etäluku ja energian kulutuksen säätö. Tällöin palveluportti tarjoaa alustan palvelun tarjoajan palveluille asiakaan kotiverkossa. Palveluporttitoteutus voi yhdistää asiakkaan laitteita, kuten jo mainitun sähkömittarin, vesimittarin tai muun älykkään laitteen, kotiverkosta ulkopuolisiin palvelun tarjoajiin, tässä tapauksessa energiayhtiöön. Kuvassa 2 on kuvattu karkeasti edellä kuvattu verkkojärjestely.



Kuva 2. Palveluportti yhdistää kodin sisäisen verkon ulkoiseen verkkoon.

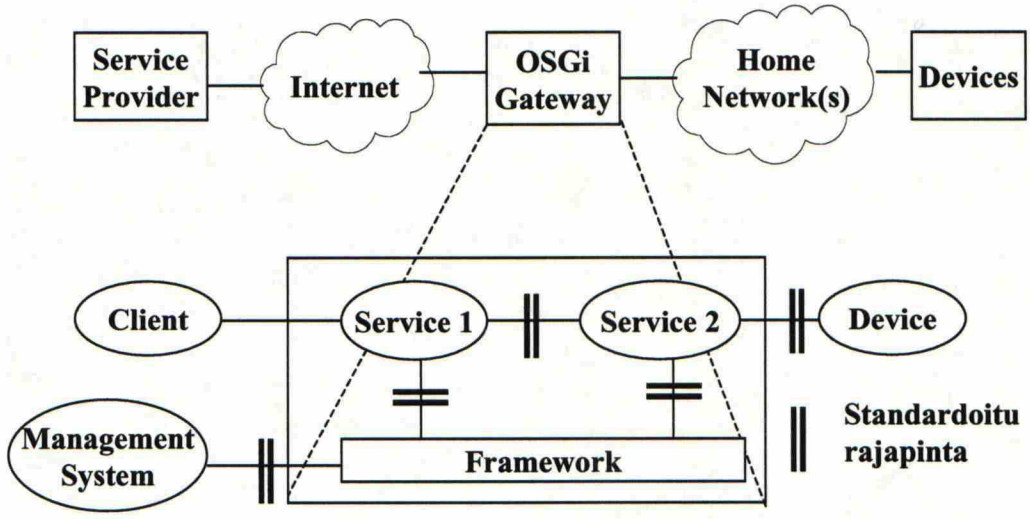
Vaikkakin palveluportti yhdistää ja hallitsee täysin uutta älykkäiden laitteiden joukkoa, tullaan tulevaisuudessa näkemään palveluportti integroituna jo olemassa oleviin laitteisiin kuten TV-sovittimiin (set-top box), kaapelimodeemeihin, hälytysjärjestelmiin jne. Vaikka kuvassa 2 on vain yksi palveluportti kotiverkon ja ulkoisen verkon välillä, OSGi:n määritelmässä tuetaan myös useiden palveluporttien, ulkoisten verkkojen liityntäpisteiden ja paikallisten verkkojen muodostamaa verkkoarkkitehtuuria.

OSGi määritelmä on suunniteltu täydentämään ja toimimaan lähes kaikkien kotiverkkostandardien ja suositusten kanssa, esimerkiksi Bluetooth, Home Audio and Video Interoperability (HAVi), Home Phoneline Networking Alliance (HomePNA), Home Radio Frequency (HomeRF), LonWorks, Jini, Universal Plug and Play (UPnP), 802.11B (Wireless LAN) ja Universal Serial Bus (USB). Vastaavasti määritelmä sisältää tuen myös useimpiin liityntäteknologioihin, kuten xDSL, WCDMA, kaapelitelevisio ja muut laajakaistaiset liityntäteknologiat. Kuvassa 3 on esitetty kuinka OSGi sijoittuu suhteessa edellä mainittuihin standardeihin. [5]



Kuva 3. OSGi ja siihen liittyvät standardit. [5]

OSGi-määritelmä kuvaa palveluportin käyttöympäristön sovellusliittymät (API). Avoimien palveluporttien tulee tukea näitä määriteltyjä sovellusliittymiä täyttääkseen OSGi-määritelmän vaatimukset (Kuva 4). Sovellusrajapinnat keskittyvät palveluiden elinkaaren, palveluiden sisäisten riippuvuuksien, tiedon, laitteiden käyttäjien ja resurssien hallintaan sekä tietoturvaan. Käyttämällä edellä mainittuja sovellusliittymiä, peruskäyttäjä voi ladata verkkopohjaisia palveluita palvelun tarjoajalta ja palveluportti hoitaa itse palveluiden asentamisen ja konfiguraation määrittämisen.



Kuva 4. OSGi:n eri standardoidut sovellusrajapinnat. [5]

1.3.1 Kodin palveluportti

Laajakaistaisten Internet-liittymien yleistyminen ja tulevien yhdistettyjen ääni-, data- ja videopalveluiden jakaminen kotiverkon eri laitteille samaa laajakaistaista yhteyttä käyttäen uskotaan tekevän kotipalveluportista avaintekijän tarjottaessa yhdistettyjä palveluita. Lukuisa määrä erityyppisiä palveluita osoittaa tarpeen useille palveluportille kodeissa. Jokaisella palveluportilla on mahdollisesti eri tapa kytkeytyä julkiseen verkkoon. Osa palveluporteista kytkeytyy palveluporttioperaattorin kautta, kun toiset palveluportit voivat käyttää hyväkseen kodissa jo olevia muita liityntäpisteitä (ADSL, kaapelimodeemi jne.).

Palveluportti (Open Service Gateway, OSG) muodostaa keskipisteen OSGi:n arkkitehtuurissa. Teknisesti palveluportti on sulautettu palvelin, joka liittää julkisessa verkossa olevat palvelun tarjoajat tai palvelun koko ajan kodin laitteiden

muodostamiin asiakkaisiin. Samalla tämä järjestely mahdollistaa tehokkaan tavan erottaa julkinen verkko kodin sisäisestä verkosta. Peruslähtökohtana palveluportissa on sen turvallisuus ja ettei se vaadi käyttäjältä ylläpitoa. Tämän lisäksi se sisältää liitännät tärkeimpiin kotiverkkojärjestelmiin esimerkiksi LON, X10 ja Ethernet.

1.4 Tutkimuksen tavoitteet

Tämän diplomityön tavoitteena on tutustua OSGi:n määritykseen sekä palveluiden tekemiseen ja hallintaan useista palvelualustoista muodostuvassa verkossa. Näitä tavoitteita täyttämään on OSGi:n määritystä tutkittaessa tarkoitus vastata seuraaviin kysymyksiin:

Onko OSGi-määritys niin valmis, että sillä voisi toteuttaa palveluita asiakkaille?

Millaisia vaatimuksia määritys asettaa palveluoperaattorille?

Työ koostuu neljästä osasta, joista ensimmäinen on teoreettinen ja käsittelee OSGi:n palvelualustamääritystä ja siinä olevia palveluita. Toisessa osassa keskitytään itse palveluihin, niiden luomiseen ja hallintaan. Kolmannessa osassa on esiteltynä esimerkkipalvelu, mikä on toteutettu yhteistyössä Siemens Metavector'in kanssa. Viimeisessä osassa käsitellään lyhyesti palveluporttiin liittyviä tieto- ja sovellusturvallisuuteen liittyviä asioita.

2. OSGI VIITEKEHYSMÄÄRITELMÄ

2.1 Oliopohjaisten ohjelmointikielten perusteet

2.1.1 Historia

Olio-ohjelmoinnin taustalla on kaksi tärkeää tietotekniikan sovellusta: simulointi ja graafiset käyttöliittymät. Ensimmäiseksi olio-ohjelmointikieleksi sanotaan Norjassa 60-luvulla kehitettyä Simula-67 ohjelmointikieltä. Vaikka kieli sisälsi jo kaikki olio-ohjelmoinnin peruskäsitteet, se ei koskaan levinnyt kovin laajaan käyttöön. Simula-67 on ollut tärkeänä esikuvana kehiteltäessä C++ -ohjelmointikieltä.

Toinen olio-ohjelmoinnin kannalta tärkeä ohjelmointikieli on 70-luvulla XEROX Parc –tutkimuslaitoksessa kehitetty Smalltalk-kieli. Kieleen liittyy oleellisena osana graafinen ohjelmointiympäristö ja valmiit luokkakirjastot, joista löytyy valmiina mm. erilaisia yleisiä tietorakenteita ja käyttöliittymien rakentamiseen tarvittavia osia.

Eräissä ohjelmointikielissä oleva moduulin (esim. Modula-2) tai pakkauksen (esim. Ada) käsite on sukua olio-ohjelmoinnille. Näissä kielissä määritellyt moduulit ja pakkaukset sisältävät oliomaisia piirteitä. Niistä kuitenkin puuttuu monia olio-ohjelmoinnille tärkeitä ominaisuuksia, kuten perintä ja dynaaminen sidonta. [6]

2.1.2 Olio-ohjelmoinnin käsitteitä

Seuraavassa on lyhyesti esiteltynä olio-ohjelmoinnin peruskäsitteitä yksinkertaisten esimerkkien avustamana.

2.1.2.1 Oliot

Olio-ohjelmoinnin perusajatuksena on esittää ohjelman käsittelemät tiedot oliona (object). Olio sisältää sekä tietoja että tietoja käsitteleviä toimintoja. Tätä tietojen ja niihin liittyvien toimintojen yhteenliittämistä kutsutaan enkapsuloinniksi (encapsulation). Monesti kirjallisuudessa puhutaan, että olio tarjoaa palveluja, joita olion asiakkaat voivat käyttää lähettämällä oliolle viestejä tai sanomia (message).

Sanomien lähettämien olio-ohjelmoinnin yhteydessä ei kuitenkaan viittaa mihinkään sanomanvälitykseen tietoliikenteen mielessä, vaan vastaa tavallista aliohjelmakutsua. Oliolle lähetettävä viesti voi sisältää erilaisia parametrejä, jotka voivat olla myös toisia olioita.

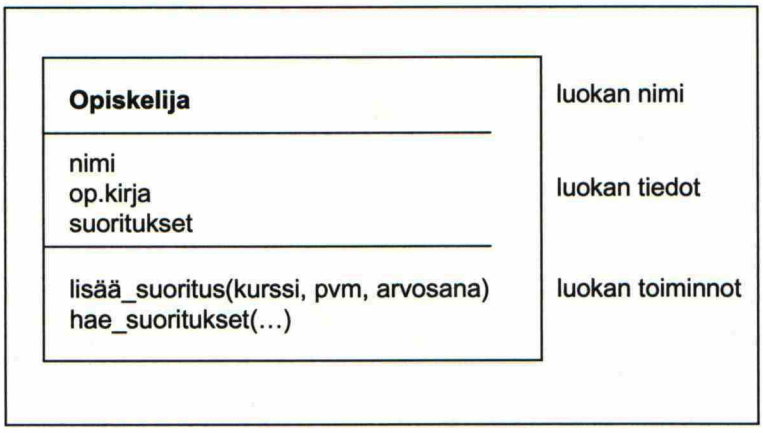
Oliokielet eroavat toisistaan sen suhteen, kuinka tiukasti ne pitävät kiinni periaatteesta, että kaikki tiedot todella ovat olioita. Tässä suhteessa esimerkiksi Smalltalk on hyvin puhdasoppinen kieli, kun vastaavasti C++ on ns. hybridikieli, joka esittää vain osan ohjelman käsittelemistä tiedoista olioina ja loput ovat kokonaan oliomaailman ulkopuolella.

2.1.2.2 Luokat

Olio-ohjelmointi perustuu olioiden lisäksi luokkiin. Jokainen olio on tällöin jonkin luokan instanssi. Esimerkiksi koulun tietojärjestelmä voi sisältää luokan `opiskelija`, jonka instanssit esittävät yksittäisiä opiskelijoita. Luokat eivät kuitenkaan ole olio-ohjelmoinnissa välttämättömiä.

Luokka määrittelee instanssiensa yhteiset tiedot. Esimerkiksi opiskelijaluokka voi kertoa, että jokainen opiskelijaolio sisältää opiskelijan nimen, opintokirjan numeron ja luettelon opintosuorituksista. Opiskelijaluokka voi myös määritellä toiminnon `lisää suoritus`, joka kohdistuu opiskelijaolioon ja jolle on annettava parametreinä kurssikoodi, päivämäärä ja arvosana.

Monesti oliopohjaisiin suunnittelumenetelmiin liittyy graafisia merkintöjä. Seuraavassa kuvassa (Kuva 5) oleva kaavio on yksi mahdollisista graafisista esitystavoista, joilla edellä kuvattu opiskelijaluokka voitaisiin kuvata.



Kuva 5. Luokkakaavio

2.1.2.3 Tiedon piilottaminen

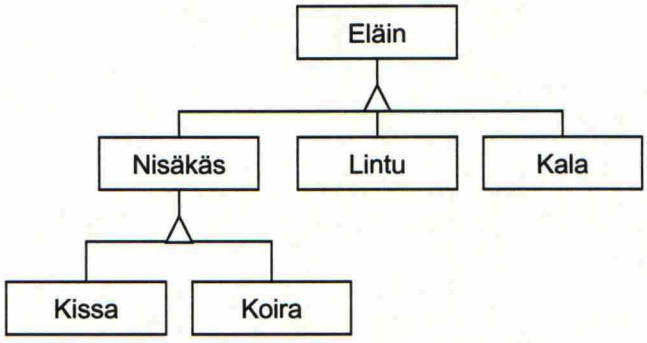
Luokkaan liittyy monesti menetelmä tiedon piilottamiseksi (data hiding). Tällä tarkoitetaan luokan ominaisuuksien, tietojen ja toimintojen, jakamista kahteen osaan. Jotkut ominaisuuksista näkyvät ja ovat käytettävissä kaikkialla luokan ulkopuolella. Joissain oliopohjaisissa ohjelmointikielissä näitä ominaisuuksia kutsutaan luokan julkiseksi osaksi. Toiset luokan ominaisuudet ovat vastaavasti vain luokan itsensä käytettävissä eivätkä näy luokan ulkopuolelle. Nämä ominaisuudet muodostavat luokan yksityisen osan.

Luokan julkinen osa määrittelee luokan rajapinnan eli sen, mitä luokan instansseilla voi tehdä, ja luokan yksityinen osa sisältää näiden toimintojen toteutukseen tarvittavat asiat. Luokan asiakkaiden ei tarvitse välittää yksityiseen osaan tehdyistä muutoksista niin kauan kun luokan julkinen rajapinta säilyy muuttumattomana. Usein kaikki luokan tallentamat tiedot määritellään yksityisiksi, jolloin niihin pääsee käsiksi vain luokan julkisten toimintojen kautta.

Edellä mainitun lisäksi on myös olemassa toisenlaisia näkyvyyssääntöjä. Osa rajapinnasta ja/tai ominaisuuksista voi olla julkisia esimerkiksi saman pakkauksen (esim. Javan pakkaus) luokille, mutta muilta piilotettuja. Näkyvyyttä voidaan jaotella myös esimerkiksi sen perusteella, onko asiakas ko. luokan aliluokka (ks. perintä).

2.1.2.4 Perintä

Enkapsuloinnin ohella olio-ohjelmoinnin tärkeä periaate on luokkien välinen perintäsuhde. Oliokielessä perintä liittyy yleensä kiinteästi luokkahierarkian käsitteeseen. Otetaan esimerkki yksinkertaisesta luokkahierarkiasta, jossa perintää on merkitty OMT-tekniikan mukaisesti kolmiolla (Kuva 6).



Kuva 6. Luokkahierarkia

Kaaviosta nähdään eläinten luokkahierarkia, eläin on nisäkäs, lintu tai kala. Vastaavasti nisäkäs voi olla kissa tai koira. Monesti puhutaan yli- ja aliluokista. Esimerkiksi `nisäkäs` on luokan `eläin` aliluokka (subclass) ja luokan `kissa` yliluokka (superclass).

Luokka perii (inherit) yliluokkansa ominaisuudet. Esimerkiksi jos eläimillä on ominaisuus `paino`, niin tämä ominaisuus on myös kaikilla nisäkkäillä ja niin ollen myös kaikilla koirilla. Joskus mainitaan yli- ja aliluokkien sijasta kantaluokat (base class) ja johdetut luokat (derived class). Näillä määrittelyillä kuvan 6 `eläin` on kantaluokka ja luokasta `nisäkäs` on johdettu luokat `kissa` ja `koira`. Kaikki toiminnot, jotka on käytettävissä kantaluokan instansseille, ovat siis käytettävissä myös johdettujen luokkien instansseille.

Perintä johtaa luokkien korvaavuuteen. Joka paikassa, missä ohjelma haluaa käsitellä esimerkiksi tyyppiä `nisäkäs` olevaa oliota, todellinen olio voi olla jokin tästä luokasta johdetun luokan instanssi. Joissain ohjelmointikielissä johdettu luokka voi haluttaessa piilottaa kantaluokan ominaisuudet, jotka eivät enää ole käytettävissä kantaluokan instansseille. Tästä syntyy helposti ongelmia, minkä takia tätä ei tulisi käyttää.

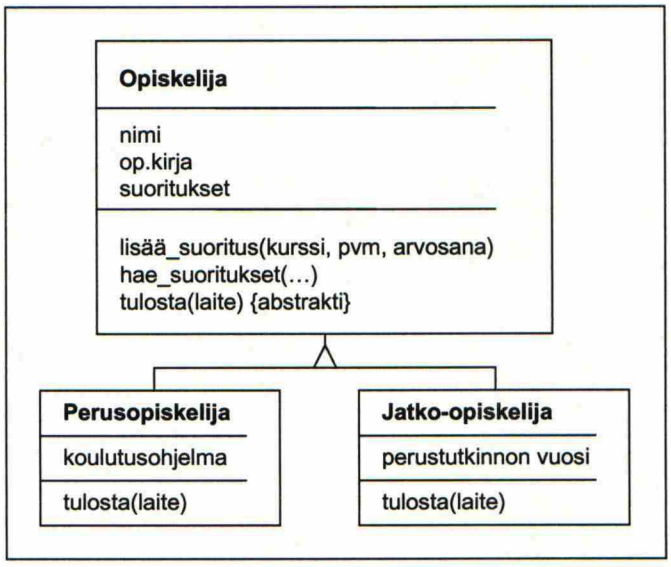
Kun luokalla voi olla vain yksi välitön kantaluokka, luokista syntyy puumainen hierarkia. Tätä kutsutaan yksinperinnäksi, koska luokka voi periä ominaisuudet vain yhdeltä kantaluokalta. Moniperinnässä luokalla voi olla useampia välittömiä kantaluokkia. Tästä tarkemmin kohdassa Moniperintä.

Luokkahierarkiassa voi olla joko abstrakteja tai konkreettisia luokkia. Nämä luokkatyypit on helppo selittää pienen esimerkin avulla. Kuvassa 6 luokat `eläin` ja `nisäkäs` voisivat olla abstrakteja luokkia, kun taas luokat `kissa` ja `koira` ovat konkreettisia. Tämä tarkoittaa sitä, että olio ei voi olla pelkästään eläin tai nisäkäs, vaan esimerkiksi nisäkkään täytyy olla joko kissa tai koira. Olioinstansseja voi tehdä vain konkreettisista luokista. Abstraktia luokkaa käytetään vain kantaluokkana toisten luokkien johtamiseen. Konkreettinen luokka voi myös toimia kantaluokkana, mutta tyypillisesti konkreettiset luokat ovat luokkahierarkian lehtiä, eikä niistä enää johdeta uusia luokkia.

2.1.2.5 Polymorfismi

Polymorfismi eli monimuotoisuus tarkoittaa olio-ohjelmoinnissa sitä, että samanniminen toiminto tarkoittaa eri asiaa eri luokkien (yhteydessä oleville) instansseille.

Jatketaan hieman aikaisemmin käytettyä esimerkkiä koulun tietojärjestelmästä. Johdetaan oppilasrekisteriin kaksi uutta luokkaa: `perusopiskelija` ja `jatko-opiskelija`. Samalla `opiskelija`-luokkaan voidaan lisätä toiminto `tulosta`, joka tulostaa opiskelijan tiedot sopivassa muodossa jollekin tulostuslaitteelle. Jotta perusopiskelijan ja jatko-opiskelijan tiedot voitaisiin tulostaa eri muodossa, tulee uusiin johdettuihin luokkiin lisätä tulostustoiminto. Polymorfismi syntyy siitä, että toiminto lisätään myös kantaluokkaan `opiskelija`. Kantaluokassa tulostustoiminto määritellään kuitenkin olevan abstrakti toiminto, jolloin opiskelijaluokka ei anna mitään tulostusta tulostustoiminnolle. Opiskelijaluokka vain määrittelee, että kaikille opiskelijoille on käytettävissä tulostustoiminto, mutta jättää abstraktista luokasta johdetun konkreettisen luokan tehtäväksi toteuttaa toiminto. Tilanne on kuvattu kuvassa 7.



Kuva 7. Abstraktin toiminnon toteutus konkreettisissa luokissa.

Instansseja voi tehdä vain konkreettisista luokista ja konkreettisissa luokissa on toteutettava kaikki kantaluokan abstraktit toiminnot. Abstrakteja toimintoja voi siis olla vain abstrakteissa luokissa. Voidaan myös sanoa, että luokasta tulee abstrakti, jos siinä on vähintään yksi abstrakti toiminto.

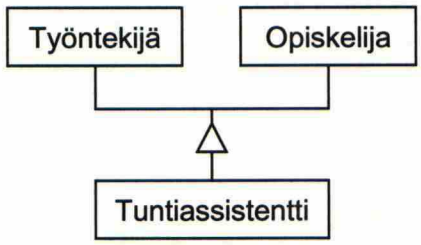
Polymorfismiin liittyy toimintojen dynaaminen sidonta. Oletetaan, että ohjelmassa on olio, jonka tiedämme olevan jonkinlainen opiskelija, mutta emme tiedä, onko hän perus- vai jatko-opiskelija. Kun tähän olioon kohdistetaan ohjelmassa tulostustoiminto, ohjelma suorittaa automaattisesti oikean tulostustoiminnon eli siinä luokassa määritellyn tulostustoiminnon, jonka instanssia ollaan tulostamassa.

Kantaluokka voi myös toteuttaa toimintoja niin, että johdettu luokka voi halutessaan korvata toteutuksen omalla tavallaan. Esimerkkinä tästä voisi kuvata tilannetta, jossa tietojärjestelmässä täytyy vielä esittää luokka `erityisopiskelija`, jonka instanssien halutaan käyttäytyvän lähes samoin kuin luokan `perusopiskelija` instanssien. Uusi luokka voitaisiin johtaa perusopiskelijaluokasta niin, että uudessa luokassa määritellään vain ne toiminnot, joiden täytyy toimia eri tavoin kuin perusopiskelijaluokassa olevien toimintojen.

2.1.2.6 Moniperintä

Moniperinnässä luokalla voi olla useampia välittömiä kantaluokkia. Jatkaen esimerkeissä kouluaiheista linjaa voidaan ajatella luokkaa `tuntiassistentti`, jolla

on kantaluokkina sekä `opiskelija`- että `työntekijä`-luokka (Kuva 8). Moniperintä tarkoittaa, että tuntiassistentti on samanaikaisesti sekä opiskelija että työntekijä. Luokan `tuntiassistentti` instanssiin voi toisin sanoen kohdistaa kaikki luokissa opiskelija ja työntekijä määritellyt toiminnot.



Kuva 8. Moniperintä

Ongelmana moniperinnässä on nimikonfliktien hallinta. Jos molemmissa kantaluokissa määriteltäisiin saman niminen toiminto, kuinka silloin hallitaan tilanne, jossa sitä kutsutaan `tuntiassistentti`-luokasta? Tähän kysymykseen ei ole yksikäsitteistä oikeaa vastausta ja olio-ohjelmointikielissä onkin valittu erilaisia ratkaisuja. Ohjelmointikieli voi esimerkiksi vaatia, että tuntiassistenttiluokan määrittelyssä määritellään kumpaa perittyä toimintoa käytetään.

Yllä oli selostettu toteutuksen ja ominaisuuksien moniperintää. Moniperintää voidaan kuitenkin jäljitellä myös rajapinnoilla, joissa kuvataan vain julkinen osuus. Tällöin luokka periytyy korkeintaan yhdestä luokasta, mutta voi toteuttaa useita rajapintoja.

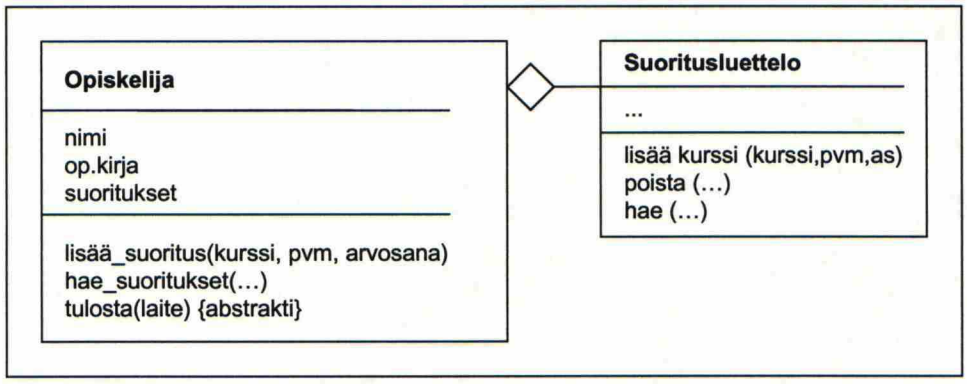
2.1.2.7 Komponentit

Perintä on olio-ohjelmoinnissa ensiarvoisen tärkeä suhde kahden luokan välillä. Perinnän ohella luokkien välillä voi olla muitakin suhteita. Yksi yleinen virhe olio-ohjelmoinnissa on ajatella, kahden luokan omatessa yhteisiä piirteitä, että yhden luokan tulisi aina periä toinen luokka. Perinnän sijasta luokka voi esimerkiksi olla toisen luokan komponentti.

Jos luokka B on luokan A komponentti, jokainen luokan A instanssi (voi) sisältää yhtenä osana luokan B instanssin. Palataan takaisin opiskelijarekisteriesimerkkiin. Koska suoritusluettelon käsittelyyn liittyy erilaisia sääntöjä, on selkeämpää muodostaa oma luokka `suoritusluettelo`, joka sisältää luetteloon liittyvät toiminnot

`lisää`, `poista` jne. Tämän jälkeen voidaan määritellä suoritusluettelosta opiskelijaluokan komponentti, eli kukin `oppilas`-olio sisältää yhden `suoritusluettelo`-olion.

`Opiskelija`luokka sisältää edelleen samat suoritusluetteloon liittyvät julkiset toiminnot, mutta toimintojen toteutus on muuttunut. Esimerkiksi opiskelijaluokan toiminto `lisää_suoritus` ei ole muuta kuin kutsu opiskelijan komponenttina olevan suoritusluettelon toiminnolle `lisää_kurssi`. Tiedon piilottamisen ansiosta opiskelijaluokan asiakkaiden ei tarvitse tietää toteutuksen muutoksesta. Uusi opiskelijaluokka ja sen komponentti on esitetty kuvassa 9. Viiva luokkien välillä kuvaa yhteyttä luokkien välillä ja kärjellään seisova neliö viivan päässä kertoo suoritusluettelon olevan opiskelijaluokan komponentti.



Kuva 9. Komponenttisuhde luokkien välillä.

2.1.3 Olioperusteinen ohjelmointitapa

Olioperustaisuus lienee varteenotettavin haastaja aiemmin vallalla olleelle proseduraaliselle eli osittelevalle ohjelmointitavalle. Osittava ohjelmointisuunnittelu toteuttaa yleisiä modulaarisuuden periaatteita tarkentamalla toimintoja aliohjelmien ja perusrakenteiden kautta aina ohjelmointikielen käskyjen tasolle. Etuna tällä ajattelutavalla on, että sovelluksen kehittäminen etenee melko nopeasti suunnittelusta toteutukseen.

Osittelevan ohjelmointitavan puutteina voidaan mainita ainakin se, ettei se nojaudu kohdemaailman tietomalliin. Tämä tietomalli kuvaa pysyvämpää rakennetta kuin toimintamalli ja aiheuttaa siten todennäköisemmän tarpeen tehdä muutoksia

ohjelmaan. Lisäksi kun muutoksia tehdään ylätason toimintoihin ne heijastuvat myös muualle ohjelmistoon. Tämä lisää muutoksen vaatimaa työn määrää. Toisaalta ohjelmiston tilan tarve kasvaa käytettäessä osittelevaa ohjelmointitapaa. Komponenttien uudelleenkäyttö on hankalaa niiden riippuvuudesta kunkin sovelluksen erityispiirteistä johtuen. Lisäksi komponentit voidaan joutua toteuttamaan kokonaan uudelleen toisaalla ohjelmistossa.

Olioperusteisen ohjelmointitavan yhtenä etuna voidaan sanoa olevan se, että se perustuu - toisin kuin ositteleva ohjelmointitapa - kohdemaailman tietokantamalliin. Siinä tunnistetut melko pysyvät objektit kuvataan olioilla, joihin liittyy sekä objektien ominaisuudet että niiden toiminnallinen puoli. Olioperusteisen ohjelman muuttaminen on myös helpompaa, koska muutostarpeet kohdistuvat paikallisesti suhteellisen itsenäisiin luokkiin ja niiden toteutukseen, eivätkä näin leviä laajemmalle ohjelmistoon kuten osittelevalla ohjelmointitavalla toteutetussa ohjelmassa. Tämä helpottaa myös virheiden välttämistä.

Ohjelmiston laajennettavuus ja komponenttien uudelleenkäyttö on sujuvaa, kun luokasta voi muodostaa aliluokkia, jotka perivät kantaluokan ominaisuudet ja toiminnot. Toisaalta olioperustainen ohjelmistonkehitys vaatii suunnittelulta enemmän kuin osittava suunnittelu, mutta hyvä suunnittelu näkyy vastaavasti parempana ohjelmistona ja alhaisempina ylläpitokustannuksina.

2.1.4 Java

90-luvun alussa Sun Microsystems'n James Gosling suunnitteli Oak-nimisen ohjelmointikielen sulautettujen järjestelmien toteuttamiseen. Kieltä kehiteltiin edelleen useita vuosia ja uudeksi tärkeäksi sovellusalueeksi otettiin Internetiin tavalla tai toisella liittyvien ohjelmien laadinta. Tämä Oak'ista kehitelty kieli sai nimekseen Java. Itse kielen määrittely julkaistiin elokuussa 1996.

Ominaisuuksiltaan Java on olio-ohjelmointikieli. Ohjelmoinnissa käytetään alusta pitäen luokkia ja niiden ilmentymiä olioita, eli kaikki tiedot ja toiminnot ovat sisällytettävä aina johonkin luokkaan. Toisaalta Javalla voidaan myös tehdä ei-olio-ohjelmia määrittelemällä kaikki piirteet luokkakohtaisiksi (static). Luokkien periytyminen tarjoaa välineet mallinnukseen ja laajojen ohjelmistojen

toteuttamiseen. Kielenä Java tarjoaa vahvan tyypityksen eli ohjelmointijärjestelmä pyrkii pitämään huolen, ettei ohjelmoija yritä sijoittaa muuttujille vääräntyyppisiä arvoja. Samalla Java-kieli tarjoaa myös välineet erilaisten poikkeustilanteiden käsittelyyn ja rinnakkaisohjelmointiin. [7]

Ohjelmointikielenä Java on täsmällisesti määritelty, kaikkien tietotyyppien esitystavat ovat kiinnitetty, laskentatarkkuus on määritelty ja kielen valmiiden välineiden kirjastot ovat määrätty. Näiden toimenpiteiden tarkoituksena on, että missä tahansa laadittu ohjelma toimisi täsmälleen samoin kaikkialla. Itse Java-kielinen ohjelma käännetään koneriippumattomalle binääriselle välikielelle tavukoodiksi (bytecode) ja ohjelma suoritetaan tulkitsemalla tätä välikielistä ohjelmaa. Ohjelman tulkinta välikielisestä ohjelmasta luo Javan niin sanotun alustariippumattomuuden.

Java-kielen etuna alustariippumattomuuden lisäksi voidaan mainita mahdollisuus käyttää tiedon esittämiseen useimpien maailman kielien merkkejä. Tämä on mahdollista koska Java käyttää Unicode-koodausta merkkitietojen koodaamiseen.

Java tallentaa kaikki oliot muistialueelle, johon se tekee viittaukset ohjelmasta. Näitä olioihin kohdistuvia viittauksia tutkimalla Java pystyy vapauttamaan muistialueita automaattisesti. Ohjelman ajon aikana taustalla on käynnissä eräänlainen automaattinen roskienkerääjä, joka tutkii, ovatko muistialueiden varaukset käyneet tarpeettomiksi.

Tavallisten ohjelmien ja sovellusten lisäksi Java-kielellä voi laatia www-sivuille sijoitettavia sovelmia (applet). Sovelmien toiminta perustuu seuraavanlaiseen toimintoketjuun, jossa aluksi www-selain noutaa välikielisen ohjelmatiedoston palvelun tarjoajan palvelimelta. Tämän jälkeen selaimen oma tulkki suorittaa välikielisen ohjelman käyttäjän koneessa. Turvallisuussyistä ohjelman oikeudet ovat tavallista sovellusta pienemmät, esimerkiksi tiedostojen lukeminen tai kirjoittaminen ei yleensä ole mahdollista. Tätä turvatoimintoa kutsutaan Javan hiekkalaatikkomalliksi (Sand Box Model). [8]

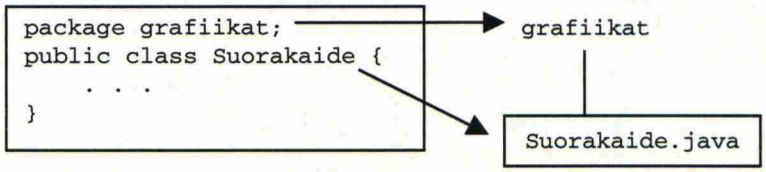
Tulevaisuudessa on mahdollista, ettei ohjelmia enää osteta, vaan niistä maksetaan käytön mukaan. Tällaisessa mallissa ohjelman osat, eli luokkakirjastot, sijaitsevat

jollain Internetiin kytketyllä palvelimella. Kun jotakin palvelua tarvitaan, se käydään hakemassa sieltä, mistä se löytyy. Vastaavasti haettu ohjelma voi tarvittaessa hakea omia lisäosiaan Internetistä ja niin edelleen. Ohjelmia ei näin tarvitse ostaa omaksi vaan, joka käyttökerrasta maksetaan erikseen.

Edellä kuvattu tulevaisuuden malli on jo osittain käytössä avoimen palveluportin (open service gateway, OSG) yhteydessä. Juuri Javan mahdollistama alustariippumattomuus sekä komponenttien hallinta ja lataaminen tarvittaessa ovat olleet avaintekijöitä valittaessa ohjelmointiympäristöä OSGi-ympäristöön.

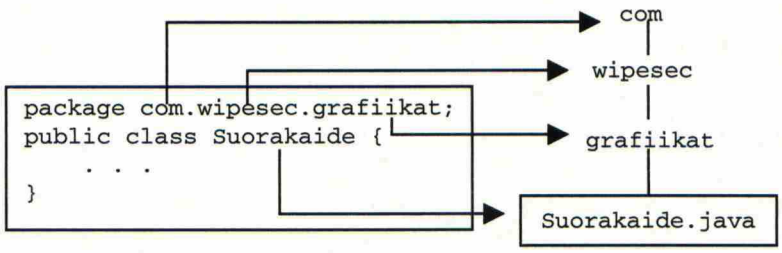
2.1.5 Luokkien nimeäminen

Useat Java-alustan toteutukset perustuvat hierarkkiseen tiedostojärjestelmään hallittaessa lähdekoodia ja luokkatiedostoja. Tämä tapahtuu seuraavasti: luokan tai rajapinnan lähdekoodi on tallennettu tekstitiedostoon, jonka nimi on luokan tai rajapinnan nimi ja tarkennin on `.java`. Tämän jälkeen lähdekooditiedosto laitetaan hakemistoon, joka viittaa sen pakkauksen nimeen, johon luokka tai rajapinta kuuluu. Esimerkiksi `Suorakaide`-luokan lähdekoodi on tallennettu tiedostoon nimeltä `Suorakaide.java` ja tiedosto sijaitsee hakemistossa `grafiikat` (Kuva 10). Itse `grafiikat`-hakemisto voi sijaita vapaasti tiedostojärjestelmässä.



Kuva 10. Suorakaide luokan nimeäminen ja sijoittaminen.

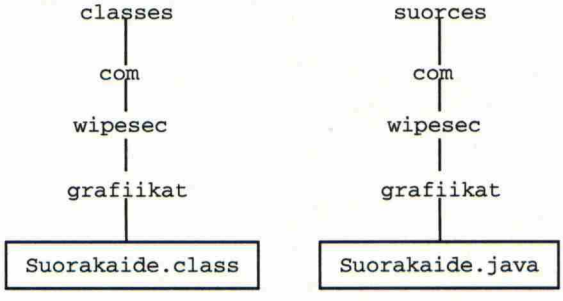
Yleisesti on tapana, että yritykset käyttävät käänteistä Internet-verkkoalueen nimeä pakkauksensa nimissä. Esimerkiksi kuvitteellinen yritys Wipesec, jonka Internet-verkkoalueen nimi on `wipesec.com`, liittää kaikkien tuottamiensa pakkauksiensa nimen eteen `com.wipesec`. Jokainen osa pakkauksen nimessä vastaa alihakemistoa. Näin Wipesec'n tuottama `grafiikat`-pakkaus, joka sisältää `Suorakaide.java` lähdekooditiedoston, sisältää kuvassa 11 esitetyn sarjan alihakemistoja.



Kuva 11. Pakkauksen `com.wipesec.grafiikat` kuvaama alihakemistojärjestelmä.

Kun lähdekoodi käännetään suoritettavaksi ohjelmaksi, kääntäjä luo jokaiselle luokalle ja rajapinnalle oman tulostiedoston. Tulostiedoston nimi on edelleen sama kuin lähdekooditiedoston, eli luokan tai rajapinnan nimi, mutta tarkennin on `.class`.

Kuten `.java`-tiedoston tulee `.class`-tiedoston olla omalla paikallaan alihakemisto järjestelmässä. Alla olevassa kuvassa 12 on vertailtu `.java`- ja `.class`-tiedostojen hakemistorakennetta. Tämän hakemiston ei kuitenkaan tarvitse olla sama kuin missä lähdekooditiedostot ovat. Tämä mahdollistaa sen, että käännetyt luokkatiedostot voidaan toimittaa eteenpäin ilman, että itse lähdekoodia tarvitsee paljastaa.



Kuva 12. `.class`- ja `.java`-tiedostojen hakemistorakenteen vertailu.

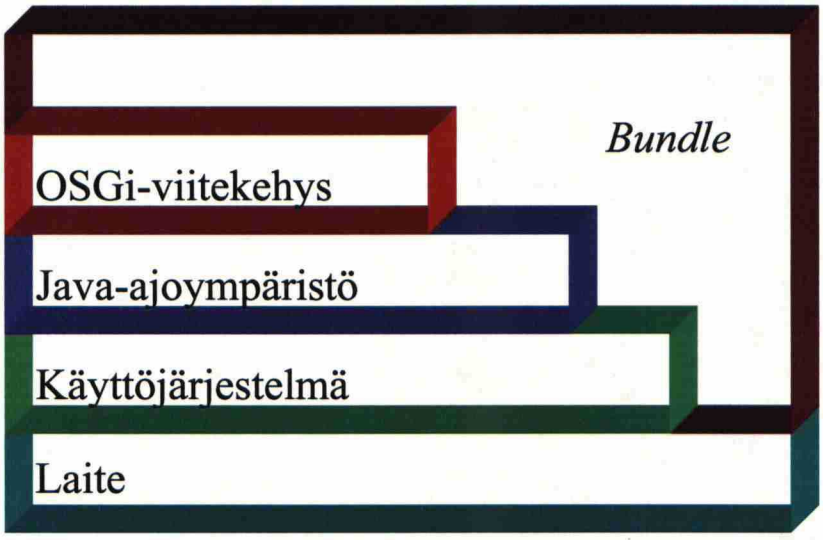
Miksi luokkien ja rajapintojen nimeäminen pitää tehdä näin hankalasti? Yksinkertainen syy tähän on lähdekoodi- ja luokkatiedostojen hallinta niin, että kääntäjä ja tulkki löytävät kaikki ohjelmiston käyttämät luokat ja rajapinnat. Kun kääntäjä kohtaa uuden luokan kääntäessään ohjelmaa, sen pitää pystyä löytämään luokka selvittääkseen luokan nimen ja muut luokaan liittyvät tiedot. Vastaavasti kun tulkki suorittaa ohjelmaa sen tulee voida löytää luokka, jotta se voi käyttää luokan toteuttamia metodeja. Hakemisto, mistä sekä kääntäjä että tulkki aloittavat

luokkatiedoston hakemisen määritellään kyseisten ohjelmistojen asennuksen yhteydessä. Esimerkiksi Windows-ympäristössä tämä määritellään luokkapolkulistauksessa (class path). [9]

2.2 Viitekehys

Viitekehys (framework) muodostaa OSGi-palvelualustamääritelmän ytimen. Se tarjoaa yleiskäyttöisen, turvallisen ja hallitun Java-viitekehyksen, joka tukee laajennettavia ja ladattavia palvelusovelluksia *bundle'eita*. Nämä *bundle'it* ovat pieniä, itselatautuvia sovelluksen osia. OSGi-yhteensopiva laite voi ladata, asentaa ja poistaa tarvittaessa OSGi-*bundle'eita* automaattisesti ilman käyttäjän toimintaa. Asennettu *bundle* voi rekisteröidä useita palveluita, joita voidaan jakaa viitekehyksen hallinnoimina toisten *bundle'ien* käyttöön.

Kuvassa 13 on havainnollistettu palveluportin ohjelmistorakennetta. *Bundle* voi antaa toimintopyyntöjä suoraan jokaiselle kerrokselle tai OSGi-viitekehyksen kautta. Käytettäessä suoria pyyntöjä *bundlen* toteutus ei enää ole täysin alustariippumaton, kuten se olisi käytettäessä OSGi-viitekehystä toiminnepyyntöjen välittäjänä. Viitekehyksen välittäessä pyyntöjä OSGi-viitekehys ja Javan käyttöympäristö kääntävät pyynnön käyttöjärjestelmän ja laitteen ymmärtämiksi pyynnöiksi. [10]



Kuva 13. Palveluportin ohjelmistorakenne tasoina.

Viitekehys käsittelee dynaamisesti ja sopeutuvasti *bundle'ien* asennuksen ja päivittämisen sekä riippuvuudet palveluiden ja OSGi-ympäristöön asennettujen

toisten *Bundle'ien* välillä. Viitekehys tarjoaa *bundle*-ohjelmoijalle tarvittavat resurssit Javan alustariippumattomuuden ja dynaamisen koodin lataamisen hyväksikäyttämiseen, kun kehitellään laajoja palveluita pienellä muistilla varustettuihin laitteisiin.

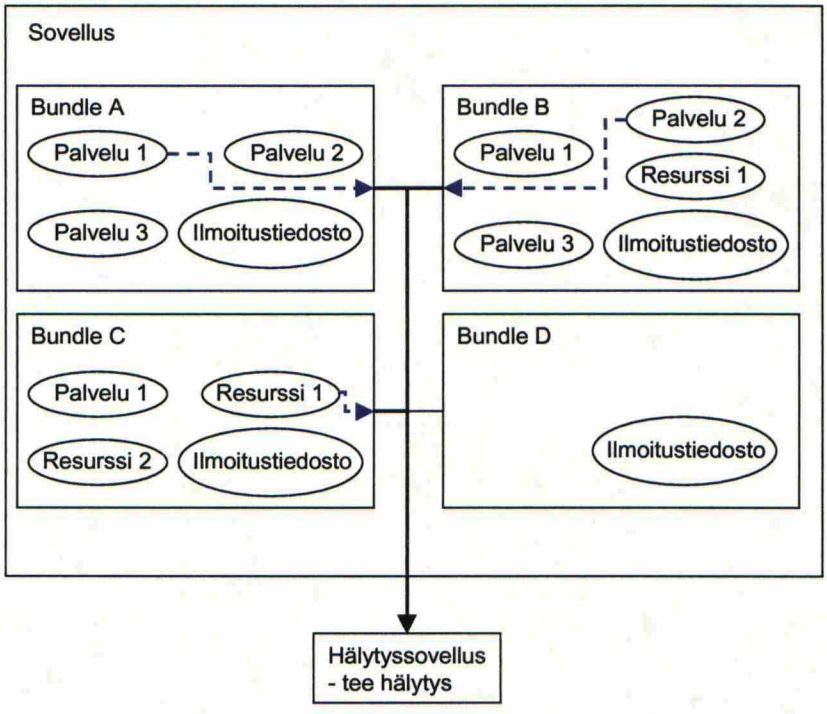
Vähintään yhtä tärkeä viitekehyksen ominaisuus on sen tarjoama yhdenmukainen ja suppea ohjelmointimalli. Tämä ohjelmointimalli yksinkertaistaa palvelun kehitystä ja käyttöönottoa erottamalla palvelun määrityksen (Java-rajapinta) palvelun toteutuksesta. Valinta tietyn toteutuksen tai tietyn palvelutoimittajan välillä voidaan tehdä vasta palvelua käytettäessä.

Yhdenmukainen ohjelmointimalli auttaa *bundle*-ohjelmoijaa hallitsemaan palvelun sopeutuvuuteen liittyviä asioita. Ne ovat viitekehyksen toiminnan kannalta tärkeitä, koska viitekehys on tarkoitettu toimimaan vaihtelevia ominaisuuksia omaavissa laitteissa. Erilaiset laitteisto-ominaisuudet voivat vaikuttaa monin tavoin palvelun toteutukseen. Yhdenmukaiset rajapinnat varmistavat sen, että ohjelmistokomponentteja voidaan sekoittaa ja yhdistellä ja silti tuloksena on stabiili järjestelmä. Otetaan esimerkiksi palvelu, joka on tarkoitettu ajettavaksi korkeatasoisessa laitteessa, jossa palvelu voi tallentaa tietoa laitteen massamuistiin esimerkiksi paikalliselle kovalevylle. Kun samaa palvelua ajetaan laitteessa, jossa ei ole massamuistia, pitää palvelun tallentaa tarvitsemansa tieto muualle esimerkiksi kotiverkon sisällä toiseen laitteeseen. Sovelluskehittäjät, jotka käyttävät tätä palvelua, voivat ohjelmoida omat *bundle'insa* käyttämään määriteltyjä rajapintoja ilman, että heidän tarvitsee huolehtia sitä, mihin palvelutoteutukseen *bundle* sijoitetaan.

Viitekehys sallii *bundle'ien* valita sopiva toteutus tarjolla olevista toteutuksista ajon aikana (runtime) viitekehyksen palvelurekisteristä (service registry). *Bundle'it* rekisteröivät uusia palveluita, vastaanottavat tiedonantoja palveluiden tilasta tai etsivät olemassa olevia palveluita sopeutuakseen laitteen suorituskykyyn ja ominaisuuksiin. Uusia *bundle'eita* asentamalla voidaan lisätä laitteen tai sovelluksen ominaisuuksia. Olemassa olevia *bundle'eita* voidaan myös muokata ja päivittää ilman että järjestelmää tarvitsee uudelleen käynnistää. Viitekehys tarjoaa mekanismeja tukemaan tätä tyyppiesimerkkiä, mikä auttaa *bundle*-suunnittelijaa käytännössä ohjelmoimaan mukautuvia *bundle'eita*.

Jokainen palvelu toteuttaa osan koko toteutuksesta. *Bundle'it* voidaan ladata laitteeseen vasta kun *bundle'in* toteuttamaa palvelua tarvitaan. Esimerkkinä voidaan käyttää myöhemmin tarkemmin esiteltävää palvelua, joka koostuu liikkeen havainnoinnista, hälytyksestä ja sähköpostin lähettämisestä. Käyttäjä aktivoi viitekehyksen avulla toteutetun hälytysjärjestelmän. Tällöin hälytyssovellus ilmoittaa viitekehykselle tarpeen liikkeenhavainnoinnista. Viitekehys lataa ja aktivoi oikeat *bundle*'it, jotka toteuttavat halutun palvelun. Tämän jälkeen liiketunnistimien havainnoima tila on hälytysjärjestelmän valvonnassa. Kun liikeilmaisimet havaitsevat liikettä, ilmoittaa liikkeen havainnointi siitä edelleen hälytysjärjestelmälle, joka ryhtyy tarvittaviin toimenpiteisiin ja lähettää käyttäjälle viestin hälytyksestä sähköpostilla. Vastaavasti sähköpostipalvelun lataaminen voi tapahtua dynaamisesti vasta tarvittaessa.

Kuvassa 14 on yksinkertainen esimerkki, missä sovellus koostuu neljästä *bundle'ista*. Esimerkkipalveluna on hälytyksen tekeminen. Sovellus käyttää *bundle'eissa* A ja B sijaitsevia kahta palvelua ja *bundle'issa* C olevaa resurssia (merkitty katkoviivalla). Samalla tavoin on mahdollista jakaa palveluita ja resursseja myös toisten sovellusten kesken.



Kuva 14. Esimerkki OSGi-viitekehyksen sovelluksen rakenteesta.

Käytännössä viitekehys voidaan jakaa kolmeen osa-alueeseen: palveluihin, *bundle'eihin* ja *bundle*kontekstiin. Palvelut ovat Java-luokkia, jotka tuottavat tiettyjä toiminnallisuuksia. Palvelut ovat yleensä toteutettu siten, että rajapinta ja toteutus ovat erillään. *Bundle* on vastaavasti toiminnallinen yksikkö, jolla palveluita siirretään. *Bundle*konteksti on *Bundle'ien* suoritusympäristö viitekehyksessä.

2.3 Bundle'it

OSGi-ympäristössä *bundle'it* ovat ainoita olioita (entity), mihin Java-pohjaisia sovelluksia sijoitetaan. *Bundle* sisältää Java-luokkia sekä muita resursseja, jotka yhdessä voivat tarjota toimintoja peruskäyttäjälle sekä komponentteja muille *bundle'eille*. Näitä *bundle'in* tarjoamia komponentteja kutsutaan palveluiksi. Käytännössä *bundle'it* on sijoitettu Java-arkistotiedostoihin (Java Archive, JAR). JAR-tiedostoja käytetään sovellusten tallentamiseen standardoidussa ZIP-pohjaisessa Java-tiedostomuodossa.

Bundle JAR-tiedosto sisältää resurssit, joilla voidaan toteuttaa useita palveluja. JAR-tiedostossa olevien resurssien ei kuitenkaan ole pakko toteuttaa yhtään palvelua. Resurssit voivat olla Java-ohjelmointikielen luokkatiedostoja, HTML-tiedostoja, ohjesivuja, ikoneja tms. Lisäksi JAR-tiedostossa on ilmoitustiedosto (manifest), jolla kuvataan JAR-tiedoston sisältö sekä *bundle'in* tiedot. Ilmoitustiedostossa käytetään hyväksi otsikkorakennetta määrittelemään viitekehyksen tarvitsemat tiedot *bundle*'in asennukseen ja aktivointiin. [10]

Bundle'in sisältämässä JAR-tiedostossa määritellään riippuvuudet muihin resursseihin kuten Java-paketteihin, joiden pitää olla asennettuna ennen kuin *bundle* voidaan aktivoida. Viitekehyksen tulee selvittää *bundle'in* ilmoitustiedostossa määritellyt vaatimukset ennen *bundle'in* aktivointia.

Bundle'issa on myös tieto erityisestä luokasta, joka määritetään toimimaan *bundle*-aktivoijana (bundle activator). Ennen kuin *bundle* voidaan asentaa, on *bundle*'in asennusluokkan oltava asennettuna, jotta asennettavan *bundle*'in käynnistys- ja pysäytys-metodit ovat viitekehyksen käytettävissä. *Bundle'in* toteutus edelläkuvatulla tavalla mahdollistaa sen, että *bundle* määrittää itselleen alkuarvoja

palvelua rekisteröitäessä sekä poistaa *bundle'in* tarjoamien palveluiden tiedot pysäyttämisen yhteydessä.

Edellä mainittujen tietojen lisäksi JAR-tiedoston `OSGi-OPT` hakemistossa tai yhdessä sen alihakemistoista voi olla lisädokumentaatiota. Kuitenkin niin, ettei mitään näissä tiedostoissa olevaa tietoa voida vaatia *bundle'ia* käytettäessä, vaan sitä käytetään esimerkiksi *bundle'in* lähdekoodin taltiointipaikkana. Hallintajärjestelmät usein poistavat lisädokumentaation säästääkseen tallennustilaa OSGi-ympäristössä.

2.3.1 Ilmoitustiedosto

Bundle voi sisältää itseään kuvaavaa tietoa ilmoitustiedostossa, joka on liitetty *bundle'in* JAR –tiedostoon nimellä `META-INF/MANIFEST.MF`. Viitekehyksessä on määritelty ilmoitustiedoston sisältämät kentät. Näitä ovat mm. `Export-package`, joka määrittelee *bundle'in* tarjoamat paketit ja `Bundle-Activator`. Näitä kenttiä käyttämällä *bundle*suunnittelijat voivat antaa tarkentavaa tietoa *bundle'ista.* Ilmoitustiedoston rakenne määritellään tarkemmin Sun:in julkaisemassa JAR File Specification julkaisussa. [11] Kaikki ilmoitustiedoston kentät ovat vapaaehtoisia eikä kentille ole määritelty `Bundle-Classpath` –kenttää lukuun ottamatta oletusarvoa.

Viitekehyksen tulee aina käsitellä vähintään ilmoitustiedoston yleinen osuus. Toteutuskohtaiset tiedot viitekehys voi jättää käsittelemättä. Viitekehys jättää käsittelemättä myös sellaiset kentät, joita ei ole määritelty ilmoitustiedostossa. Suunnittelija voi lisätä omia kenttiä tarpeen mukaan ilmoitustiedostoon. Jos ilmoitustiedoston kenttä sisältää tuntemattoman arvon, ei sitä käsitellä vaan arvo hylätään. Näin voi käydä, jos kenttään on vahingossa syötetty väärä arvo.

Alla on esimerkki Gatespacen tuottaman OSGi ympäristöön kuuluvan Messenger-viestipalvelun *bundle'in* ilmoitustiedostosta. Tiedostosta käy selville *bundle'in* toimittaja ja mitä sen on tarkoitettu tekevän. `Bundle-Activator` –kenttä nimeää bundle'in luokan, joka toteuttaa *bundle'in* aktivoijarajapinnan. `Export-service` – kenttä määrittää mitä palveluita *bundle* voi rekisteröidä viitekehyksen palvelurekisteriin. Tätä kenttää ei kuitenkaan käytetä viitekehyksessä, vaan lähinnä palvelinpuolen hallintatyökaluilla. Vastaavasti `Import-service` määrittää ne palvelut mitä ko. *bundle* voi käyttää. Samoin kuten edellinen kenttä, ei tätäkään käytetä

hyväksi viitekehyksessä vaan lähinnä palvelimen hallintatyökaluilla. `Import-Package` -kenttä määrittää ne paketit, jotka *bundle* tarvitsee. Näiden pakettien tulee olla jonkun toisen *bundle*'in asennuksen yhteydessä tarjoamia.

```
Manifest-Version: 1.0
Bundle-Vendor: Gatespace AB
Bundle-Version: 2.0
Bundle-Category: service
Bundle-Activator: com.gatespace.bundle.messenger.Messenger
Bundle-Copyright: 'Copyright (c) 1999-2001 Gatespace AB. All Rights Re
 served.'
Bundle-Config: !/config.xml
Bundle-DocURL: http://www.gatespace.com/doc
Created-By: 1.2.2 (Sun Microsystems Inc.)
Import-Service: com.gatespace.service.log.LogService, org.osgi.service
 .log.LogService, com.gatespace.service.messenger.MessageTransporter,
 com.gatespace.service.messenger.AddressConverter, com.gatespace.servi
 ce.messenger.MessageProtocol,
Bundle-Name: messenger
Bundle-ContactAddress: info@gatespace.com
Export-Service: com.gatespace.service.messenger.MessengerService, com.
 gatespace.service.console.CommandGroup, org.osgi.service.cm.ManagedSe
 rvice
Bundle-Description: The Gatespace Messenger service bundle
Import-Package: com.gatespace.service.console,com.gatespace.service.lo
 g,com.gatespace.service.messenger,com.gatespace.utils,org.osgi.framew
 ork,org.osgi.service.cm
```

2.3.2 Järjestelmäbundle

Tavallisten *bundle'ien* lisäksi viitekehys itsessään on esitetty järjestelmä*bundle'ina*. Järjestelmä*bundle'in* avulla viitekehys voi rekisteröidä palvelut, joita muut *bundle'it* voivat käyttää.

Järjestelmä*bundle* eroaa tavallisista *bundle'eista* mm. tunnukseltaan. Järjestelmä*bundle'in* tunnus viitekehyksen *bundle*listauksessa on aina nolla (0) (Kuva 15). Elinkaarihallintamalli ei vastaa täysin tavallisen *bundle'in* elinkaarihallintamallia. Järjestelmä*bundle'in* kanssa `start`-metodi ei ole käytettävissä, koska viitekehyksen ollessa käynnissä on myös järjestelmä*bundle* käynnistetty.

Kuva 15. Viitekehykseen asennettujen bundle'ien listaus, mistä selkeästi huomataan järjestelmäbundle'in tunnus "0".

Vastaavasti `uninstall`-metodia ei voida käyttää, koska Järjestelmä*bundle'ia* ei voi poistaa viikehyksestä. `Stop`-metodi pysäyttää viitekehyksen ja `update`-metodi pysäyttää ensin viitekehyksen ja käynnistää sen sitten uudelleen. Tarkemmin *bundle'in* elinkaarihallintamallista kerrotaan kohdassa *Bundle*olio.

2.3.3 Hallintabundle

OSGi viitekehyksen määritelmä tarjoaa mekanismeja, muttei toimintaperiaatteita. Toimintaperiaatteet, esimerkiksi mistä *bundle'it* ladataan, toteutetaan niin sanotuilla hallinta*bundle'eilla*. Hallinta*bundle'ella* on oikeus suorittaa viitekehyksessä hallinnollisia toimintoja ja tarjota vaaditut toimintamallit. Hallinta*bundle* on tavallinen *bundle* ja sen tulee noudattaa kaikkia *bundle'eita* koskevia sääntöjä. OSGi:n viitekehyksen määritelmä ei määrittele kuinka ensimmäinen hallinnointi*bundle* asennetaan viitekehykseen, koska tämä saattaa vaihdella eri käyttöönottotilanteiden välillä. Tämä malli sallii operaattorin määritellä tarvittavat toimintaperiaatteet yksinkertaisella ja johdonmukaisella tavalla.

2.3.4 Bundle'ien nimeäminen

Kun luokka ladataan Javan virtuaaliympäristöön (Java VM), se tehdään erillisen luokkalataajan (`ClassLoader`-olio) avulla. Luokan viitatessa toiseen luokkaan tai resurssiin löytyvät nämä viitatut luokat saman luokkalataajan avulla. Luokkalataaja

voi ladata luokan itse tai se voi siirtää luokan latauksen toiselle luokkalataajalle. Tämänkaltainen lähestyminen luo tehokkaasti luokkien nimiavaruuden. Tämän nimiavaruuden avulla luokka on tunnistettavissa täysin luokitellun (fully qualified) nimen ja nimen luoneen luokkalataajan perusteella. Edellä kuvattu järjestely viittaa siihen, että luokka voidaan ladata viitekehykseen useita kertoja eri luokkalataajien kautta.

Jokaiseen *bundle'iin*, joka on asennettu ja analysoitu viitekehyksessä, tulee olla liitettynä luokkalataaja. Yhteen *bundle'iin* voi kuitenkin olla liitettynä useita luokkalataajia. Luokkalataaja tarjoaa *bundle'ille* oman nimiavaruuden päällekkäisten nimien välttämiseksi, sekä mahdollistaa pakettien jakamisen toisten *bundle'ien* kanssa. *Bundle'in* luokkalataajan täytyy löytää *bundle'issa* olevat luokat ja resurssit etsimällä *bundle'in*-luokkapolusta (classpath), joka määritellään *bundle'in* ilmoitustiedostossa.

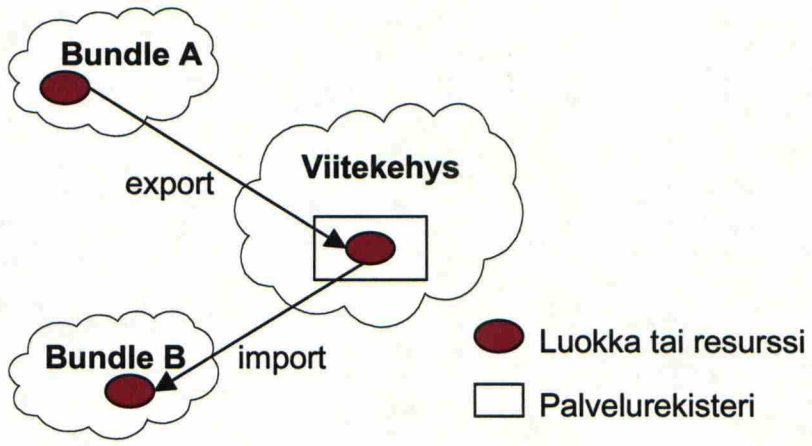
Bundle'it tekevät yhteistyötä jakamalla objekteja, jotka ovat keskenään sovitun luokan tai rajapinnan instansseja. Tämä luokka pitää ladata molemmille *bundle'eille* käyttäen samaa luokkalataajaa, muuten jaetun olion käyttö aiheuttaa luokkajakopoikkeaman (ClassCastException). Välttääkseen edellä kuvatun poikkeaman viitekehyksen tulee varmistaa, että kaikki luokkien tuojat (importer) viedyssä (export) paketissa käyttävät samaa luokkalataajaa ladatessaan luokan tai rajapinnan.

Esimerkiksi *bundle* voi rekisteröidä palveluolion *class.com.acme.C* alle käyttäen viitekehyksen palvelurekisteriä. On tärkeää että *bundle*, joka luo palveluolion (*bundle* A) ja *bundle*, joka hakee sen palvelurekisteristä (*bundle* B) jakavat saman luokan *class.com.acme.C*, mistä palveluolio tulee instantioida. Jos *bundle* A ja B käyttävät eri luokkalataajaa ladatessaan luokan *class.com.acme.C.*, *bundle'in* B yritys jakaa palveluolio omaan versioon luokasta *class.com.acme.C* aiheuttaa poikkeaman `ClassCastException`.

2.3.5 Pakettien jakaminen

Bundle voi tarjota kaikkia siihen kuuluvia luokkia ja resursseja määrittelemällä pakkauksen nimen Export-Package kohtaan *bundle'in* ilmoitustiedostossa. Kullekin

viitekehykseen vietäväksi tarjotulle pakkaukselle viitekehys valitsee *bundle*'in. Tämä *bundle* toimii luokkien ja resurssien toimittajana muille viitekehykseen asennetuille *bundle'eille*. Siinä tapauksessa, että viitekehyksessä useampi *bundle* tarjoaa samaa pakettia, viitekehys valitsee yhden joka tarjoaa paketin kaikille halukkaille *bundle*'eille. Kuvassa 16 on kuvattu yksinkertaistetussa muodossa *bundle'in* pakettien tarjoaminen (export) ja käyttäminen (import).



Kuva 16. Kaaviokuva bundle'in pakettien jakamisesta.

Valitsemalla yksi paketti kaikkien tarjoajien joukosta varmistaa sen, että kaikki *bundle'it* jakavat samat luokka- ja resurssi-määritykset. Jos *bundle* ei osallistu pakettien jakamiseen eli sen ilmoitustiedostossa ei ole Export-Package ja Import-Package kohtia, niin *bundle'in* yritykset ladata luokka tai resurssi paketista ei voi kohdistua jaetuissa pakkauksissa oleviin luokkiin ja resursseihin. Tällaisessa tilanteessa luokkia ja resursseja etsitään ainoastaan järjestelmän luokkapolusta sekä *bundle'in* JAR-tiedostosta.

Bundle'ien pakettien jako tehdään *bundle'in* analysointivaiheessa. Vain onnistuneesti analysoidut *bundle'it* voivat jakaa paketteja. Analysoimattomat *bundle'it* eivät voi osallistua mitenkään pakettien jakoon. Lisäksi *bundle'illa* on oltava riittävät oikeudet voidakseen osallistua pakettien jakamiseen.

Bundle julistaa ne resurssit, jotka se tarjoaa muiden *bundle'ien* käyttetäväksi ilmoitustiedoston Export-Package kohdassa. Vastaavasti ne resurssit, joita *bundle* tarvitsee toimiakseen, ilmoitetaan ilmoitustiedoston Import-Package kohdassa

2.3.6 Pakettien tarjoaminen

Export-Package otsikko ilmoitustiedostossa sallii *bundle'in* tarjota Java paketteja OSGi-ympäristöön asennetuille toisille *bundle'eille.*

Viitekehys varmistaa, että tarjotun paketin nimiavaruudessa olevat luokat ja resurssit ladataan paketin tarjoavasta *bundle'ista.* Jos useampi kuin yksi *bundle* julkaisee saman paketin omissa ilmoitustiedostoissaan, viitekehyksen tulee valita se *bundle,* joka tarjoaa uusimman version tarjotusta paketista.

```
Export-Package: javax.servlet;
specification-version="2.1",
javax.servlet.http;
specification-version="2.1"
```

Yllä olevassa ilmoitustiedostosta otetussa esimerkissä *bundle* tarjoaa kaikkia version 2.1 `javax.servlet` ja `javax.servlet.http` -paketteihin kuuluvia luokkia muiden viitekehykseen asennettujen *bundle'ien* käyttöön.

Vaikkakin *bundle* voi jakaa kaikki luokansa ja resurssinsa, se ei ole kuitenkaan suositeltavaa ellei kyseessä ole erityisen harvoin päivitystä tarvitsevat kirjastotietueet. Syynä edellä mainittuun varovaisuuteen on, ettei viitekehys pysty välttämättä vapauttamaan jaettujen luokkien varaamaa tilaa *bundle'ien* poiston yhteydessä.

Bundle'in rakenne, jossa palvelun toteutus ja rajapinnat ovat erillään, on suositeltava. Tällöin *bundle'in* suunnittelijan tulee laittaa rajapinnat erilliseen Java – pakettiin, joka sitten jaetaan muiden *bundle'ien* käyttöön. Samalla palvelun toteutukseen liittyvät luokat pidetään muilta *bundle'eilta* piilossa.

Jos rajapinnalla on useita toteutuksia useissa *bundle'eissa, bundle*-suunnittelija voi pakata rajapinnan kaikkiin rajapintaan liittyviin *bundle'eihin. Bundle'ia* asennettaessa viitekehys valitsee näistä *bundle'eista* yhden, joka tarjoaa rajapinnan kaikille muille *bundle'eille.* Koska muutokset rajapinnassa vaikuttavat kaikkiin siihen liittyviin toteutuksiin, tulee rajapinnan suunnittelu tehdä huolellisesti ja pitää rajapinta muuttumattomana rajapinnan julkaisemisen jälkeen.

2.3.7 Pakettien käyttäminen

Import-Package otsikko ilmoitustiedostossa sallii *bundle'in* pyytää pääsyä toisten OSGi –ympäristöön asennettujen *bundle'ien* jakamille paketeille. Viitekehys varmistaa, että analysoinnin jälkeen *bundle* näkee ainoastaan yhden version Import-Package -otsikossa mainituista paketeista.

Paketin tarjoaminen ei suoraan tarkoita sitä, että tarjoava *bundle* käyttää tarjoamiansa luokkia. Kuten jo aiemmin on mainittu, useat *bundle*'it voivat tarjota samaa pakettia ja viitekehys valitsee niistä yhden paketin tarjoajaksi.

Epäsuorasti *bundle* käyttää tarjoamansa paketin nimeä ja versiota ja näin ollen erillinen `Import-Package` -otsikko ilmoitustiedostossa on tarpeeton. Jos *bundle* voi toimia tarjoamansa paketin määritystä vanhemmalla versiolla kuin se tarjoaa, silloin se voidaan mainita ilmoitustiedostossa.

```
Import-Package: javax.servlet;
specification-version="2.1"
```

Yllä olevassa ilmoitustiedostosta otetussa esimerkissä *bundle* pyytää, että se analysoidaan vastaamaan version 2.1 tai sitä uudempaa `javax.servlet` -pakettia.

2.3.8 Bundle'in luokkapolku

Bundle'in sisäiset luokkapolkuriippuvuudet määritellään ilmoitustiedoston kohdassa `Bundle-Classpath`. Näin *bundle* voi ilmoittaa sisäisen luokkapolkunsa käyttäen yhtä tai useampaa sisältämäänsä JAR-tiedostoa. `Bundle-Classpath` on pisteellä erotettu luettelo tiedoston nimistä.

Luokkapolun riippuvuudet analysoidaan seuraavasti. Jos `Bundle-Classpath` otsikkoa ei ole määritelty ilmoitustiedostossa, niin käytetään oletusarvoisesti `Bundle-Classpath` arvona pistettä. Vastaavasti jos `Bundle-Classpath` on määritelty, eikä piste ole mukana määrityksessä, niin vain *bundle'in* JAR –tiedostossa määritetyt JAR-tiedostot etsitään eikä *bundle'in* JAR –tiedostoa itseään.

Alla on esimerkki *bundle'in* `Bundle-Classpath` otsikosta. Tässä tapauksessa *bundle* laittaa esiin kaikki omassa JAR –tiedostossa olevat luokat ja resurssit sekä `servlet.jar` -tiedostossa määritetyt luokat ja resurssit.

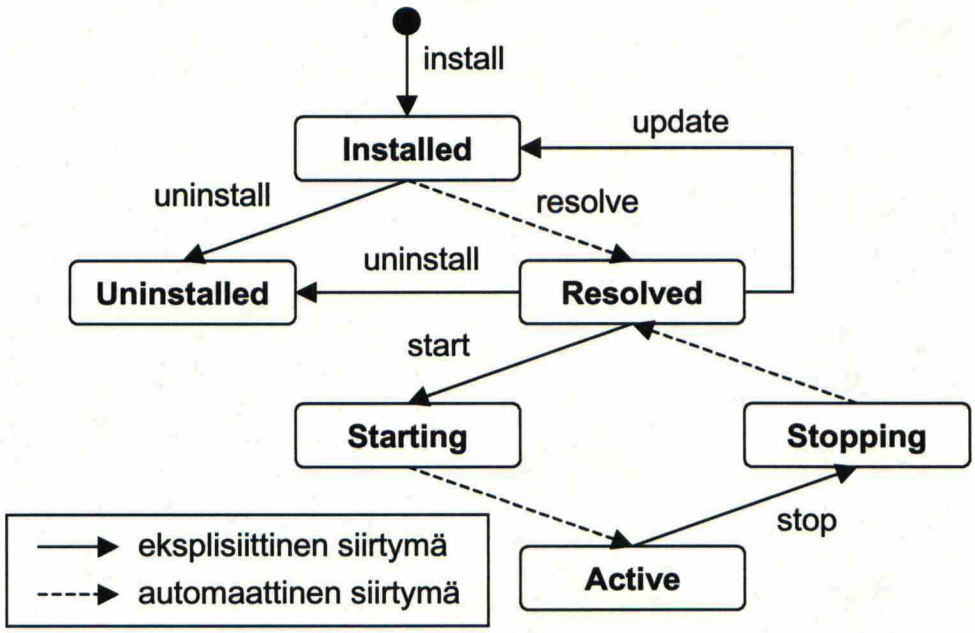
```
Bundle-Classpath: ., lib/servlet.jar
```

2.3.9 Bundleobjekti

Jokaiselle OSGi-ympäristöön asennetulla *bundle'illa* on viitekehyksessä oma *bundle*olio. Sitä voidaan käyttää *bundle'in* elinkaarenhallintaan. *Bundle,* joka vastaa kaikkien muiden *bundle'ien* elinkaaren hallinnasta kutsutaan hallinta*bundle'iksi.*

Jokaisella *bundle'illa* on oma ainutlaatuinen tunnisteensa. Tunnistetta ei tule käyttää uudelleen toiselle *bundle'ille,* vaikka sama *bundle* asennettaisiin uudelleen viitekehykseen. Samoin tunnisteen arvo ei voi muuttua *bundle'in* ollessa asennettuna tai kun *bundle* päivitetään. *Bundle*tunniste (`bundleId`) saadaan *bundle*-rajapinnan `getBundleId()`-metodilla. Metodi palauttaa *bundle'in* tunnistemääreen (Identifier attribute).

Bundle'eille on määritelty viitekehyksessä neljä eri tilaa, joista *bundle* voi olla vain yhdessä tilassa kerrallaan. Nämä neljä tilaa ovat poistettu (`uninstalled`), asennettu (`installed`), analysoitu (`resolved`) ja aktiivinen (`active`). Kun *bundle* on onnistuneesti asennettu sen tila on asennettu. Kun kaikki Java-luokkariippuvuudet on tarkistettu ja luokat ovat *bundle'in* saatavilla, on *bundle* analysoitu-tilassa. Analysoitu-tila ilmaisee, että *bundle* on valmis käynnistettäväksi tai että *bundle* on pysäytetty. Kuvassa 17 on esitetty havainnollistava tilakartta viitekehyksen tarjoamista *bundle'in* tiloista ja siirtymistä tilojen välillä. [10]



Kuva 17. Viitekehyksen tarjoamien Bundle'ien tilakartta. [10]

Käynnistymässä (`starting`) ja pysähtymässä (`stopping`) ovat molemmat eräänlaisia välitiloja *bundle'in* tilan vaihtuessa analysoidusta aktiiviseen ja vastaavasti päinvastoin *bundle'ia* pysäytettäessä. Aktiivisessa tilassa oleva *bundle* on onnistuneesti käynnistetty ja toiminnassa. Poistetussa tilassa olevaa *bundle'ia* ei voi enää siirtää toiseen tilaan.

Kun *bundle* on asennettu, se on tallennettu viitekehyksessä pysyvään tallennuspaikkaan, missä *bundle* säilyy kunnes se poistetaan viitekehyksestä. *Bundle'ia* käynnistettäessä sekä pysäytettäessä tallennetaan sen tila samaiseen pysyvään tietokantaan. *Bundle*, jonka tila on tietokantaan tallennettu aktiiviseksi tulee aina viitekehyksen käynnistyessä käynnistää. *Bundle'in* aktiivisuus merkintä poistetaan *bundle'in* pysäytyksen yhteydessä.

2.3.10 Bundle'in asentaminen

Kun *bundle* asennetaan viitekehykseen on asennus pysyvä. Tämä tarkoittaa, että *bundle* säilyy asennettuna sekä OSGi-viitekehyksessä että Java-virtuaalikoneessa (Java Virtual Machine, Java VM) kunnes se on sieltä yksiselitteisesti poistettu. Toisaalta asennus on myös jakamaton (atomic) eli *bundle* pitää asentaa kokonaisuudessaan. Asennuksen epäonnistuessa kaikki asennettavaan *bundle'iin*

liittyvät tiedot poistetaan ja viitekehys saatetaan asennusyritystä edeltäneeseen tilaan.

Kun *bundle'ia* asennetaan, viitekehys yrittää analysoida *bundle'in* alkuperäiskoodin (native code) riippuvuuksia. Jos tämä epäonnistuu, ei *bundle'ia* voida asentaa viitekehykseen. *Bundle'in* asennuksen jälkeen luodaan *bundle*–olio, minkä avulla suoritetaan loput tarvittavista elinkaaritoiminnoista.

2.3.11 Bundle'in analysointi

Bundle siirtyy analysoitu –tilaan viitekehyksen tarkastettua *bundle'in* riippuvuudet. Näitä riippuvuuksia ovat *bundle'in* luokkapolun, alkuperäiskoodin sekä paketin riippuvuudet. Tiedot edellä mainittuihin riippuvuuksiin löytyvät ilmoitustiedoston vastaavista kentistä. Poikkeuksena on paketin riippuvuustiedot, jotka sijaitsevat Import-Package ja Export-Package –kentissä.

Jos kaikki *bundle'in* riippuvuudet on onnistuneesti analysoitu, kaikki *bundle'in* ilmoitustiedoston Export-Package kentässä mainitut paketit jaetaan OSGi-ympäristön käyttöön.

2.3.12 Bundle'in käynnistäminen

Bundle rajapinta määrittelee `start()` –metodin, jolla *bundle'it* voidaan käynnistää. Jos tämä metodi suoritetaan onnistuneesti, siirtyy *bundle* käynnissä –tilaan.

Bundle'in ollessa analysoitu –tilassa, sen aktivointi tapahtuu kutsumalla *bundle'in* aktivoijaoliota (Activator object). Aktivoijaolio määrittelee metodit, jolla viitekehys herätetään *bundle'ia* käynnistettäessä ja pysäytettäessä.

Jotta viitekehykselle saataisiin tieto *bundle'in* aktivoijana toimivasta yksilöllisestä luokkaosoitteesta, määrittää *bundle*-suunnittelija sen ilmoitustiedoston `Bundle-Activator` –kentässä. Esimerkiksi `Bundle-Activator`-kenttä voi olla seuraavanlainen:

```
Bundle-Activator: com.gatespace.bundle.messenger.Messenger
```

Viitekehys instantioi olion tästä luokasta ja jakaa sen `BundleActivator`'iin. Tämän jälkeen viitekehys kutsuu `start`-metodia *bundle*'in käynnistämiseksi.

`BundleActivator`'in tarjoaminen on vapaaehtoista. Esimerkiksi kirjasto*bundle*'eille, jotka tarjoavat vain paketteja, ei yleensä tarvitse määrittää `BundleActivator`'ia. *Bundle*, joka tarjoaa palveluita, tarvitsee vastaavasti `BundleActivator`'in, koska näin *bundle* voi löytää `BundleContext`-olionsa ja saada hallinnan käynnistyessään.

`BundleActivator`-rajapinta määrittelee kaksi metodia *bundle*'ien käynnistämiseksi ja pysäyttämiseksi. Ensimmäinen metodeista on jo mainittu `start`-metodi, jolla voidaan allokoida *bundle*'in tarvirvitsemat resurssit sekä käynnistää säikeet (thread). Yleensä `start`-metodilla rekisteröidään myös *bundle*'in palvelut palvelurekisteriin. Jos näin ei tehdä, voi *bundle* tarvittaessa myöhemmin rekisteröidä palveluitaan, mikäli *bundle* on aktiivisessa tilassa.

2.3.13 Bundle'in pysäyttäminen

Toinen `BundleActivator`'in tarjoama metodi on `stop`-metodi. Tämä metodi poistaa kaikki `start`-metodin tekemät muutokset viitekehyksessä ja asettaa sen analysoitu-tilaan. Samalla metodi vapauttaa kaikki *bundle*'in varaamat resurssit. Myös kaikki pysäytettävään *bundle*'iin liitetyt säikeet pysäytetään.

On myös mahdollista, että *bundle*'in palvelut poistetaan rekisteristä käyttäen stop-metodia. On kuitenkin hyvä muistaa, että viitekehys poistaa automaattisesti kaikki *bundle*'in aktivoimisen yhteydessä sekä aktiivisessa tilassa ollessaan rekisteröimät palvelut, kun *bundle* pysäytetään. Jos `BundleActivator.start`-metodi on suoritettu onnistuneesti, niin sama `BundleActivator`-olio vastaanottaa kutsun `BundleActivator.stop`-metodilta, kun *bundle* pysäytetään.

Kaikki *bundle*'in tarjoamat paketit ovat edelleen toisten *bundle*'ien saatavilla. Tämä jatkettu pakettien tarjoaminen tarkoittaa sitä, että toiset *bundle*'it voivat suorittaa pysäytetyn *bundle*'in koodia. Tämä tulee ottaa huomioon *bundle*'ia suunniteltaessa, ettei vahingollista konfiguraatiota pääse syntymään. Yhtenä ratkaisuna edellä olevaan ongelmaan on pelkästään rajapintojen tarjoaminen. Näin voidaan estää käynnistämättömien *bundle'ien* koodin suorittaminen, koska rajapinnat eivät sisällä suoritettavaa koodia.

2.3.14 Bundle'in päivittäminen

Bundle-rajapinta tarjoaa kaksi metodia `update()` ja `update(InputStream)`, joiden avulla *bundle* voidaan päivittää viitekehyksessä. `Update()`-metodi päivittää *bundle'in* ja `update(InputStream)`-metodi päivittää *bundle'in* tietystä InputStream'ista. Päivitysprosessi tukee *bundle'in* siirtämistä uudempaan alaspäin yhteensopivaan versioon samasta *bundle*'ista.

Viitekehys pitää huolen siitä, että ainoastaan yksi versio *bundle*'in luokista on saatavilla. Jos päivitetty *bundle* tarjoaa paketteja toisten viitekehykseen asennettujen *bundle*'ien käyttöön, ei tarjottuja paketteja päivitetä. Vanhat versiot paketeista säilyvät aktiivisina kunnes viitekehys käynnistetään uudelleen tai `org.osgi.service.admin.PackageAdmin.refreshPackages`-metodia on kutsuttu.

2.3.15 Bundle'in poistaminen

Bundle'in poistamiseen viitekehyksestä *bundle*-rajapinta tarjoaa `uninstall`-metodin. Tämä metodi saa viitekehyksen ilmoittamaan siihen asennetuille *bundle*'eille *bundle*'in poistamisesta sekä asettaa *bundle*'in tilaksi poistettu. Viitekehys poistaa kaikki poistettuun *bundle*'iin liittyvät resurssit. Samalla se myös poistaa *bundle*'in viitekehyksen pysyvästä muistista. Tämän jälkeen *bundle*'ia ei ladata viitekehyksen uudelleen käynnistyksen yhteydessä.

Metodin suorittamisen jälkeen viitekehyksen tilan tulee vastata tilaa ennen *bundle*'in asentamista, lukuun ottamatta tilannetta missä *bundle* on tarjonnut paketteja ilmoitustiedostossaan ja on valittu näiden pakettien tarjoajaksi. Tässä tapauksessa viitekehys tarjoaa paketteja, kunnes se käynnistetään uudelleen tai `org.osgi.service.admin.PackageAdmin.refreshPackages`-metodia kutsutaan.

2.4 Bundlekonteksti

Viitekehyksen ja siihen asennettujen *bundle*'ien välinen suhde on toteutettu käyttämällä *bundle*konteksti-olioita. *Bundle*konteksti-olio edustaa OSGi-ympäristössä yksittäisten *bundle*'ien suoritusympäristöä ja toimii ikään kuin välipalvelimena (proxy) alla olevaan viitekehykseen.

*Bundle*konteksti-olio luodaan, kun *bundle* käynnistetään viitekehyksessä. *Bundle* voi käyttää omaa *Bundle*konteksti-oliota mm. asentaesaan uusia *bundle*'eita viitekehykseen, toisten *bundle*'ien kyselemiseen, rekisteröityjen palveluiden palveluolioiden hakemiseen ja palveluiden rekisteröimiseen. *Bundle*konteksti-olio on *bundle*'in käytettävissä kunnes *bundle* pysäytetään kutsumalla `stop(BundleContext)`-metodia. Liitteessä B esitellään tarkemmin miten *Bundle*konteksti-olio sijoittuu muihin viitekehyksen komponentteihin nähden.

2.4.1 Pysyvä tallennuspaikka

Viitekehys tarjoaa yksityisen pysyvän tallennuspaikkan jokaiselle viitekehykseen asennetulle *bundle*'ille. *Bundle*konteksti-rajapinta määrittää metodin, jonka avulla on mahdollista päästä käsiksi pysyvälle tallennusalueelle. Tämä metodi (`getDataFile(String)`) ottaa suhteellisen tiedoston nimen argumentiksi ja kääntää sen absoluuttiseksi nimeksi *bundle*'in pysyvässä tallennusalueessa ja palauttaa tiedosto-olion.

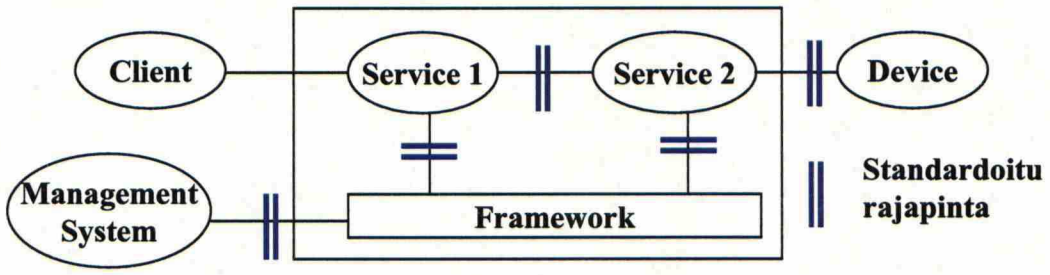
Jos pysyvä tallennusalue on annettu, antaa viitekehys *bundle*'ille tarvittavat oikeudet tallennusalueelle pääsyyn.

2.5 Palvelut

OSGi ympäristössä *bundle*'it ovat rakennettu jaetun palvelurekisterin tarjoamien yhteistoiminnassa olevien palveluiden ympärille. Nämä OSGi:n palvelut määritellään palvelurajapinnassa ja ne on toteutettu palveluobjekteina.

Palveluoliot toteuttavat yhden tai useamman palvelurajapinnan, joiden alle palvelu on viitekehyksessä rekisteröity. Palvelurajapinta määrittelee palvelun rakenteen ja toiminnan. Rajapinta tulee määritellä siten, että toteutuksesta riippuvia yksityiskohtia on mahdollisimman vähän.

OSGi on itse määritellyt valmiiksi joukon rajapintoja, joita palvelun suunnittelijat voivat käyttää. Näiden jo valmiina olevien rajapintojen lisäksi OSGi suunnittelee uusien standardoitujen rajapintojen toteuttamista viitekehyksen määritelmään. Kuvassa 18 on esitetty havainnollisesti OSGi:n määrittelemien standardoitujen rajapintojen sijainti.



Kuva 18. OSGi:n määrittelemät standardoidut rajapinnat.

Viitekehyksen ajatuksena rajapintojen standardoinnissa on, ettei jokainen palvelunkehittäjä toteuta rajapintaa omalla tavallaan ja näin lisää toimittajariippuvuutta. Kun rajapinnat ovat kaikkien käytössä, pystytään myös säästämään tallennuskapasiteettia, kun rajapintoja ei ole toteutettu jokaiseen palveluun erikseen.

Bundle'in, joka omistaa palveluolion ja jonka sisällä palveluolio suoritetaan, tulee rekisteröidä palveluolio viitekehyksen palvelurekisteriin. Näin varmistetaan palvelun saatavuus myös muille viitekehykseen asennettuille *bundle*'eille.

Viitekehys hallinnoi palvelun omistavan *bundle*'in ja palvelua käyttävien *bundle*'ien välisiä riippuvuuksia. Esimerkiksi kun *bundle* pysäytetään, poistetaan kaikki pysäytetyn *bundle*'in rekisteröimät palvelut. Tämän mahdollistaa viitekehyksen rooli riippuvuuksien hallinnassa.

Viitekehys kuvaa palvelut niiden sisäisillä palveluobjekteilla ja tarjoaa näin yksinkertaisen mutta tehokkaan kyselymekanismin. Kyselymekanismi mahdollistaa asennetun *bundle*'in pyytää ja käyttää tarvitsemiaan palveluita, jotka *bundle* itse tai jokin toinen *bundle* on tarjonnut viitekehykseen käytettäväksi. Viitekehys tarjoaa myös mekanismin, minkä avulla *bundle* voi vastaanottaa viestejä rekisteröidyistä, muutetuista tai poistetuista palveluobjekteista.

2.5.1 Palveluviiteobjekti

Normaalisti asennettuihin palveluihin viitataan palveluviiteolion kautta. Näin vältytään ylimääräisten *bundle*'ien välisten riippuvuuksien luomiselta. Palveluviite-olio voidaan tallentaa ja välittää toisille *bundle*'eille ilman, että se sisältäisi tietoja palveluiden riippuvuuksista. Kun *bundle* haluaa käyttää palvelua, lähettää *bundle*

palveluviiteolion viitekehykselle, joka hakee palvelun käyttäen `BundleContext.getService(ServiceReference)`-metodia.

Palveluviiteolio yhdistää kuvaamansa palveluolion ominaisuudet sekä muun metatiedon. Tämän metatiedon avulla *bundle* voi hakea tarvetta parhaiten vastaavan palvelun. Kun *bundle* kyselee palvelua viitekehyksen palvelurekisteristä, palauttaa viitekehys kyselevälle *bundle*'ille palveluviiteolion, jos palvelurekisteristä löytyy yksi tai useampi sopiva palvelu. Viitekehys pyrkii välttämään palveluiden viittaamisen kyseleville *bundle*'eille.

2.5.2 Palvelurajapinnat

Palvelurajapinta on määritys palvelun julkisista metodeista. Käytännössä *bundle*suunnittelija luo palveluolion toteuttamalla palvelun palvelurajapinnan ja rekisteröimällä palvelun viitekehyksen palvelurekisteriin. Palvelu voidaan ottaa käyttöön toisten *bundle*'ien toimesta rajapintanimen mukaan, kun palveluolio on rekisteröity rajapinta- tai luokkanimen alle. Näin palvelun metodeja voidaan käyttää palvelurajapinnan kautta.

Bundle rekisteröi palvelun yhden tai useamman palveluolion toteuttaman rajapinta- tai luokkanimen alle. Palveluoliota kuvaavat ominaisuudet (`Properties`), kuten toimittaja, versio jne. voidaan määritellä rekisteröintimuuttujina. *Bundle*'in pyytäessä palveluoliota viitekehyksestä voi *bundle* määrittää palvelurajapinnan, minkä kysellyn palveluolion tulee toteuttaa. Tarvittaessa *bundle* voi määritellä pyynnössään myös erityisen suodattimen tarkentaakseen hakua. Tarkemmin ominaisuuksista ja suodattimesta kerrotaan kohdassa ominaisuudet.

Monet palvelurajapinnat on määritelty samankaltaisten organisaatioiden toimesta kuten OSGi. Palvelurajapintaa, joka on hyväksytty standardiksi, voivat kaikki *bundle*suunnittelijat käyttää hyväkseen omissa toteutuksissaan.

2.5.3 Palveluiden rekisteröinti

Bundle esittelee palvelunsa viitekehykselle rekisteröimällä palveluolionsa viitekehyksen palvelurekisteriin. Rekisteröidyt palveluoliot ovat kaikkien OSGi-ympäristöön asennettujen *bundle*'ien käytettävissä. Rekisteröinnin yhteydessä

palveluoliolle voidaan määrittää joukko määrityksiä avain-arvo -parien mukaisesti. Näitä pareja hyväksikäyttäen on mahdollista tukea kehittyneitä hakumekanismeja, jotka perustuvat määreiden vertailuun.

Jokaisella viitekehykseen rekisteröidyllä palvelulla on oma yksilöllinen palvelun rekisteröinti -olio (`ServiceRegistration`) sekä yksi tai useampi palveluviiteolio (`ServiceReference`) joihin viitata. Nämä palveluviiteoliot ilmaisevat rekisteröinnin yhteydessä palveluoliolle annetut määritykset, mukaan lukien palveluolion toteuttamat palvelurajapinnat ja –luokat. Näin ollen palveluviiteoliota voidaan käyttää sellaisten palveluolioiden hakemiseen, jotka toteuttavat halutut palvelut. [10]

Viitekehys sallii *bundle*'ien rekisteröidä ja poistaa rekisteristä palveluobjekteja dynaamisesti. Siksi *bundle* voi rekisteröidä palveluolion vain `BundleActivator.start`-metodin kutsumisen ja `BundleActivator.stop`-metodin kutsumisen välisenä aikana, eli kun *bundle* on käynnistymässä, aktiivinen tai pysähtymässä.

Palvelurajapintojen nimet, joiden alle *bundle* haluaa rekisteröidä palvelunsa määritellään muuttujina `BundleContext.registerService`-metodissa. Rekisteröityä palveluoliota voidaan kuvata tarkemmin käyttämällä sanasto-oliota (`Dictionary`). Sanasto-olio sisältää kokoelman palvelun ominaisuuksia ryhmiteltyinä avain- ja arvopareihin.

Palvelurajapinnan nimet, joiden alle palveluoliot on onnistuneesti rekisteröity, lisätään automaattisesti viitekehyksen toimesta olio-luokan (`objectClass`) sisään palveluolion tietoihin. Jos *bundle* on toimittanut vastaavan tiedon se korvataan viitekehyksen tarjoamalla vastaavalla tiedolla.

Jos palveluolion asentaminen onnistuu, palauttaa viitekehys palvelun rekisteröinti –olion rekisteröivälle *bundle*'ille. Rekisteröidyn palvelun voi poistaa rekisteristä vain se *bundle*, joka omistaa palvelun rekisteröinti –olion. Näin estetään palvelun tahaton poistaminen toisen saman palvelun tarjoavan *bundle*'in poistamisen yhteydessä. Jokaisen onnistuneen palveluolion asennuksen tulee tuottaa yksilöllinen palvelun rekisteröinti –olio, vaikka sama palvelu on jo ennestään rekisteröitynä viitekehykseen useita kertoja.

Palvelun rekisteröinti –olion käyttäminen on ainoa tapa, millä pystytään luotettavasti muuttamaan palveluolion ominaisuuksia rekisteröinnin jälkeen. Pelkkä rekisteröityyn palveluolioon liittyvän sanasto-olion muuttaminen ei vaikuta rekisteröidyn palvelun ominaisuuksiin

2.5.4 Palvelun ominaisuudet

Palvelun ominaisuudet on järjestelty avain-arvo -parien mukaisesti. Ominaisuuden arvojen tulee olla luonnollinen luku tai standardoitua Java-tyyppiä, muuten muuttuja aiheuttaa ylimääräisen ei toivotun riippuvuuden *bundele*'ien välillä. Viitekehys ei pysty havaitsemaan näitä *bundle*'ien vaihtamien olioiden synnyttämiä riippuvuuksia.

Suodatinrajapinta tukee monimutkaisia suodatuksia ja näin soveltuu hyvin sopivien palveluiden hakuun. Suodatin perustuu Internet Engineering Task Force (IETF) määritykseen RFC 1960. [12] Kaikille ominaisuuksille on määritelty oma nimiavaruus viitekehyksen palvelurekisterissä. Tämän seurauksena ominaisuuksille tulee käyttää kuvaavia nimiä sekaannusten välttämiseksi. Muutamat OSGi määritelmät ovat varanneet osan ominaisuuksien nimiavaruudesta. Esimerkkinä kaikki ominaisuudet, jotka alkavat `'service.'` sekä ominaisuuksien `objectClass` ovat varattuja OSGi:n käyttöön.

2.5.5 Palvelutehdas

Palvelutehdas mahdollistaa palveluolioiden muokkaamisen *bundle*'in hakiessa palvelua `BundleContext.getService`-metodilla.

Normaalisti *bundle*'in rekisteröimä palveluolio palautetaan takaisin palvelun rekisteröivälle *bundle*'ille. Mutta jos palveluolio toteuttaa palvelutehdas-rajapinnan, kutsuu viitekehys tämän palveluolion metodeja luodakseen jokaiselle tätä palvelua käyttävälle *bundle*'ille oman yksilöllisen palveluolion. Kun *bundle* ei enää käytä palvelua esimerkiksi kun *bundle* pysäytetään, niin viitekehys ilmoittaa siitä palvelutehtaalle.

Palvelutehdasoliot helpottavat sellaisten *bundle*'ien välisten riippuvuuksien hallintaa, joita ei ole yksiselitteisesti hallittu viitekehyksen toimesta. Liittämällä palautettu palveluolio palvelua pyytäneeseen *bundle*'iin, palveluolio voi seurata

tähän *bundle*'iin liittyviä viestejä ja poistaa *bundle*'in rekisteröimiä objekteja *bundle*'in pysäyttämisen yhteydessä. Tavallisesti viestien seuraaminen ei ole tarpeellista, koska viitekehys ilmoittaa palvelutehdas-oliolle kun *bundle*'in palveluolio vapautuu.

2.5.6 Palvelun tarjoaminen ja käyttäminen

Ilmoitustiedoston `Export-Service` otsikko ilmoittaa ne rajapinnat, jotka *bundle* voi rekisteröidä. Ilmoitustiedostossa on myös lisätietoja rajapinnoista.

```
Export-Service: com.gatespace.service.messenger.MessengerService,
com. gatespace.service.console.CommandGroup,
org.osgi.service.cm.ManagedService
```

Yllä oleva esimerkki sallii *bundle*'in rekisteröidä kolme Gatespace'in tuottamaa rajapintaa `MessengerService`, `CommandGroup` sekä `ManagedService`.

Vastaavasti `Import-Service` otsikko määrittää ne rajapinnat, joita *bundle* voi käyttää. Samoin kuin `Export-Service` otsikon kohdalla, tarjoaa ilmoitustiedosto lisäinformaatiota. Esimerkissä sallitaan *bundle*'in käyttää lokipalvelua, viestipalvelua, osoitteen muunnospalvelua sekä viestiprotokollapalvelua.

```
Import-Service: com.gatespace.service.log.LogService,
org.osgi.service .log.LogService,
com.gatespace.service.messenger.MessageTransporter,
com.gatespace.service.messenger.AddressConverter,
com.gatespace.service.messenger.MessageProtocol
```

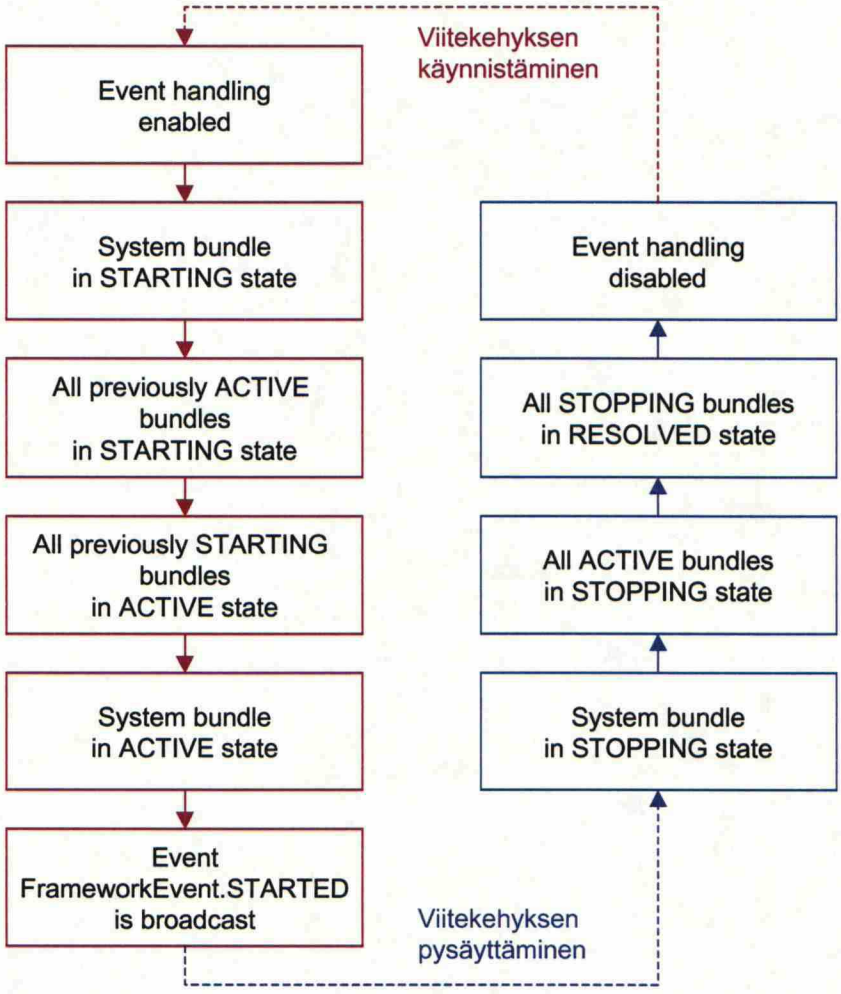
2.5.7 Palvelun poistaminen palvelurekisteristä

Palvelun rekisteröinti (`ServiceRegistration`)–rajapinta määrittää `unregister`-metodin palveluolioiden poistamiseksi rekisteristä. `Unregister`-metodi poistaa palveluolioon liittyvän palvelurekisteriolion palvelurekisteristä. Tämän jälkeen tähän palvelurekisteriolioon liittyvää palveluviiteoliota ei voida käyttää palveluolion hakemiseen.

`Unregister`-metodin sijoittaminen palvelurekisteriolioon varmistaa sen, että vain se *bundle* jonka hallussa palvelurekisteriolio on, voi poistaa siihen liitetyn palveluolion. Palveluolion rekisteristä poistavan *bundle*'in ei kuitenkaan tarvitse olla sama *bundle*, joka palveluolion alunperin on viitekehykseen rekisteröinyt. Esimerkiksi rekisteröinyt *bundle* on siirtänyt palvelurekisteriolion toiselle *bundle*'ille. Samalla vastuu palveluolion poistamisesta rekisteristä siirtyy uudelle *bundle*'ille.

2.5.8 Viitekehyksen käynnistäminen ja pysäyttäminen

Viitekehystoteutus täytyy käynnistää ennen kuin yhtään palvelua voidaan tuottaa. Itse viitekehysmääritelmä ei ota kantaa kuinka viitekehys yksityiskohtaisesti käynnistetään ja voi näin vaihdella toteutusten mukaan. Viitekehys voi käynnistyessään ottaa vastaan parametrejä komentoriviltä tai kaikki käynnistykseen tarvittavat parametrit voivat olla erityisessä konfiguraatiotiedostossa, josta ne tarpeen mukaan luetaan. Vain käynnistykseen liittyvät toiminnot ovat määritelty viitekehysmäärityksessä ja näin kaikkien toteutusten tulee toimia niiden mukaisessa järjestyksessä.



Kuva 19. Viitekehyksen käynnistäminen ja pysäyttäminen vuokaaviona.

Viitekehyksen käynnistäminen ja pysäyttäminen voidaan kuvata kuvassa 19 esitetyllä vuokaaviolla. Ensimmäisenä käynnistetään tapahtumien (event) hallinta. Tämä mahdollistaa tapahtumien välittämisen *bundle*'eille.

Viitekehyksessä on kolme tapahtumatyyppiä `ServiceEvent`, `BundleEvent` ja `FrameworkEvent`. `ServiceEvent` ilmoittaa palveluiden rekisteröinneistä, poistoista ja palveluolioiden ominaisuuksien muutoksista. `BundleEvent` ilmoittaa *bundle*'in tilan muutoksesta elinkaarimallissa. Viimeinen `FrameworkEvent` ilmoittaa viitekehyksen käynnistymisestä tai viitekehyksen kohtaamista virheistä.

Kun tapahtumien hallinta on mahdollista, siirtyy järjestelmä*bundle* käynnistymässä-tilaan. Kuten aiemmin mainittiin pitää viitekehys huolen siitä, että kaikkien asennettujen *bundle*'ien tila pysyy samana viitekehyksen uudelleen käynnistymisestä huolimatta. Tämän takia kaikki aiemmin aktiivisena olleiden *bundle*'it käynnistetään ja niiden tilat muuttuvat käynnistymässä-tilaan. Kun viitekehykseen asennetut *bundle*'it ovat saavuttaneet ennen viitekehyksen pysäyttämistä olleen tilansa, asennettu, analysoitu tai aktiivinen, siirtyy järjestelmä*bundle* aktiivinen-tilaan ja viitekehys on käynnissä. Ilmoituksena viitekehyksen käynnistymisestä lähetetään `FrameworkEvent.STARTED`-tapahtuma viitekehyksen tapahtumahallintaan.

Viitekehyksen pysäyttäminen alkaa järjestelmä*bundle*'in siirtymisellä pysähtyy-tilaan. Tämän siirtymisen voi laukaista esimerkiksi järjestelmä*bundle*'in pysäytyskäsky. Kun kaikki viitekehyksessä aktiivisena ollevat *bundle*'it ovat pysäytetty ja analysoitu-tilassa voidaan tapahtumien hallinta pysäyttää.

Seuraavassa on esitelty muutama viitekehyksen palvelu esimerkkeinä siitä, millaisia palvelut ovat sekä havainnollistamaan viitekehyksen toimintaa.

2.6 Palveluesimerkkejä

Kuten jo aiemmin on mainittu, muodostuvat palvelut pienemmistä kokonaisuuksista. Seuraavaksi käsitellään muutaman viitekehyksen toiminnan kannalta tärkeän palvelun rakenne ja toiminta. Nämä ovat pakettien hallintapalvelu, palveluiden seuranta ja käyttäjien hallinta. Jokainen mainituista palveluista on otettu mukaan OSGi-palvelualustan määritykseen ja niiden esittäminen helpottaa kokonaisuuden ymmärtämistä siirryttäessä lähemmäksi käytäntöä.

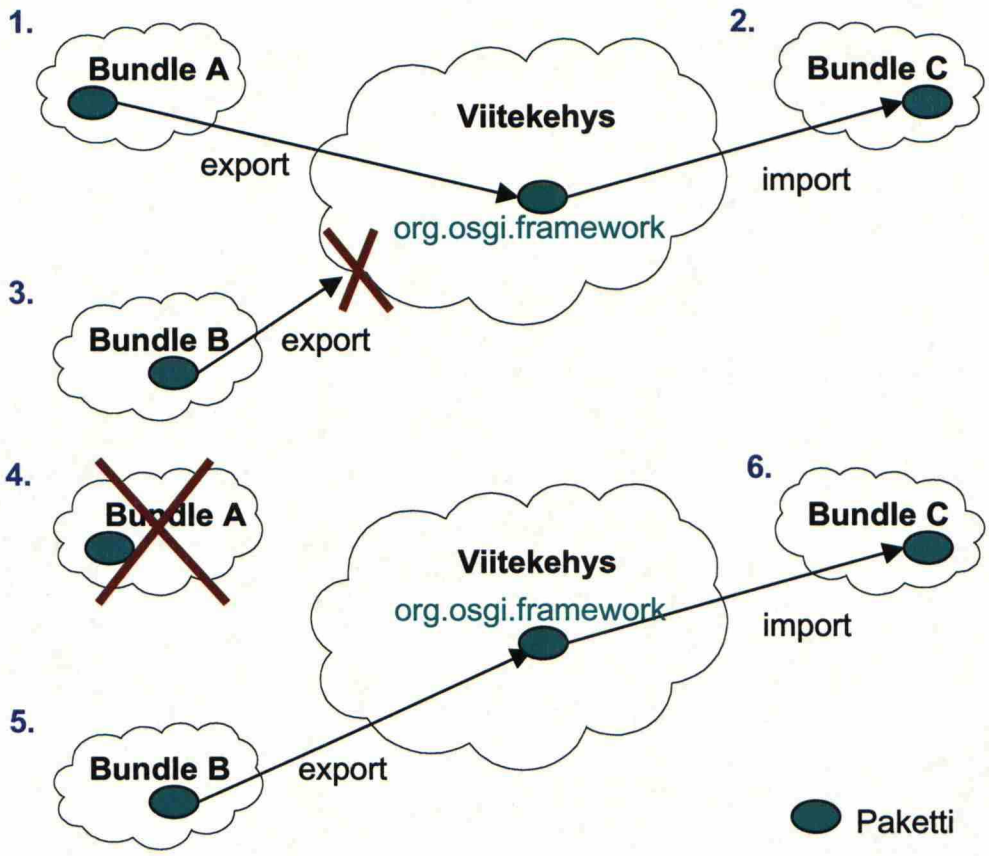
2.6.1 Pakettien hallintapalvelu

Kuten jo kohdassa Pakettien jakaminen mainittiin, voivat *bundle*'it jakaa paketteja toisten *bundle*'ien käyttöön. Pakettien jakaminen luo riippuvuuden paketin jakaneen ja pakettia käyttävän *bundle*'in välille. Tämä riippuvuus korostuu etenkin siinä vaiheessa, kun paketin jakanut *bundle* päivitetään tai poistetaan. Tällöin tulee tehdä päätös myös jaetun paketin päivittämisestä tai poistamisesta.

Viitekehyksen ensimmäisessä versiossa jätettiin päätös jaettujen pakettien päivittämisestä tai poistamisesta toteutuksen tehtäväksi. Tämä johti jaettujen pakettien päivityksen kahtiajakautumiseen välittömiksi ja viitekehyksen uudelleen käynnistyksen yhteydessä päivittäviksi. Ensimmäinen ryhmä irrotti jaetut paketit sekä pysäytti ja analysoi uudelleen niistä riippuvat *bundle*'it aina, kun paketin jakanut *bundle* päivitettiin tai poistettiin. Toinen ryhmä ei päivittänyt paketteja vaan jätti ne toisten *bundle*'ien käytettäväksi, kunnes viitekehys käynnistettiin uudelleen ja ylimääräiset paketit poistuivat.

Viitekehyksen toisen version suunnittelussa todettiin kuitenkin, että tämä oli liian tärkeä osa alue ollakseen toteutuksesta riippuva. Tämän vuoksi se otettiin viitekehyksen toisen version määrittelyyn mukaan. Versiossa 2 tämä on toteutettu erillisellä paketin hallintapalvelulla (Package Admin). Palvelu mahdollistaa hallinta*bundle*'in määritellä erilaisia toimintaperiaatteita pakettien jakamiselle.

Kun viitekehykseen asennettu *bundle* päivitetään tai poistetaan, jättää viitekehys suorittamatta *bundle*'in pakettien tarjoamiseen liittyvän valvonnan. Tämän tehtävän hoitaa hallinta*bundle*, joka voi valita pakotetun päivityksen. Tässä *bundle*'ien riippuvuussuhteet viitekehyksessä päivitetään välittömästi käyttäen pakettien hallintapalvelua. Samoin toimintamalli, missä aina käytetään viimeisimpiä viitekehyksessä tarjolla olevia paketteja, on mahdollista toteuttaa pakettien hallintapalvelun avulla. Tällöin valvotaan viitekehystapahtumista *bundle*'ien päivityksiä ja poistoja ja päivitetään näihin *bundle*'eihin liittyvät paketit käyttäen pakettien hallintapalvelua. [13]



Kuva 20. Tarjottujen pakettien hallinta

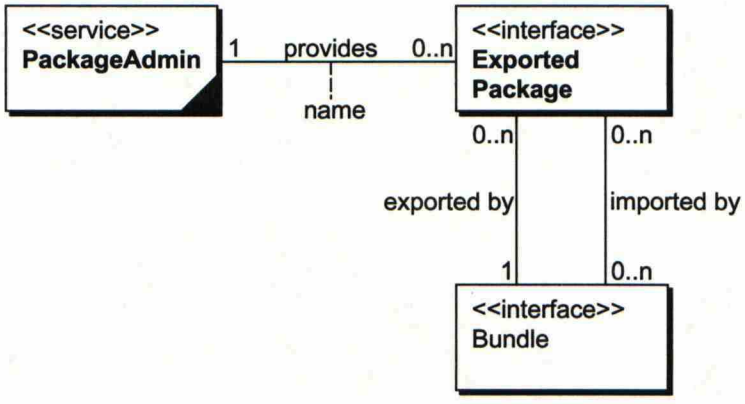
Toimintamalli on yksinkertaistetussa muodossa esitettynä kuvassa 20. Bundle A tarjoaa pakettia `org.osgi.framework` toisten viitekehykseen asennettujen *bundle*'ien käyttöön (1.). Kun *bundle* C asennetaan, haluaa se käyttää samaista pakettia (2.). Näin on syntynyt riippuvuus suhde A:n ja C:n välille. Myöhemmin viitekehykseen asennetaan Bundle B, joka myös tarjoaa samaista pakettia (3.). Tarjotun paketin version ollessa sama ei viitekehys hyväksy Bundle B.n tarjoamaa pakettia.

Kun Bundle A poistetaan viitekehyksestä (4.), poistetaan kaikki siihen liittyvät paketit. Nyt hallinta*bundle* voi päättää suoritetaanko tietojen päivitys samantien vai vasta viitekehyksen päivityksen yhteydessä. Todennäköisesti tämä tieto on määritelty hallinta*bundle*'in toteutusvaiheessa ja näin ollen toimintaperiaate säilyy muuttumattomana hallinta*bundle*'in elinkaaren ajan. Tässä esimerkissä käsitellään välitön päivitys. Kun Bundle A on poistettu, tutkii viitekehys muiden *bundle*'ien tarjoamat paketit ja havaitsee Bundle B:n tarjoavan saman paketin. Viitekehys määrittää tämän *bundle*'in paketin tarjoajaksi (5.) ja paketin hallinta*bundle* päivittää Bundle C:n tiedot muuttunutta tilannetta vastaavaksi (6.).

Tietojen päivittämisen jättäminen viitekehyksen uudelleen käynnistämiseen eroaa edellä kuvatusta hieman. Tällöin viitekehys automaattisesti käynnistymisensä yhteydessä käynnistää kaikki ennen viitekehyksen pysäyttämistä aktiivisena olleet *bundle*'it. Tällöin myös pakettien riippuvuudet tarkastetaan ja kaikilla tarjotuilla paketeilla on olemassa aktiivinen isäntä*bundle*.

Paketin hallintapalvelu on järjestelmäpalvelu, jota voidaan helposti käyttää väärin, koska se tarjoaa pääsyn viitekehyksen sisäisiin tietorakenteisiin. Tämän takia vain luotettavat *bundle*'it voivat käyttää `ServicePermission[GET]`-metodia paketin hallinta rajapinnassa. `ServicePermission[GET]`-metodi mahdollistaa sen, että *bundle* havaitsee palvelun ja ottaa sen käyttöönsä. Yleensä tämä metodi on rajattu vain hallinta*bundle*'ien käyttöön.

Pakettien hallintapalvelu koostuu seuraavista kuvassa 21 esitellyistä kokonaisuuksista. Paketinhallinta (PackageAdmin) on se rajapinta, joka tarjoaa pääsyn viitekehyksen sisäiseen pakettien jakomekanismiin. Tarjottu paketti (Exported Package) antaa tietoja paketista ja sen yhteiskäytön (sharing) tilasta. Kuvassa esitetty *bundle* voi olla esimerkiksi hallinta*bundle*, jonka avulla operaattori voi toteuttaa operaattorikohtaisia sääntöjä pakettien tarjoamiseen. [13]



Kuva 21. Paketin hallintapalvelun rakenne. [13]

2.6.2 Palveluiden seuranta

Viitekehys tarjoaa tehokkaan ja dynaamisen ohjelmointiympäristön, missä *bundle*'eja voidaan asentaa, poistaa, käynnistää ja pysäyttää ilman, että itse viitekehystä tarvitsee pysäyttää. Viitekehys tarkkailee b*undle*'ien välisiä

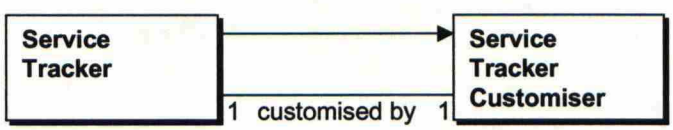
riippuvuuksia, mutta *bundle*'ien on oltava yhteistyössä viitekehyksen kanssa, jotta riippuvuuksia voidaan käsitellä oikein.

Eräs tärkeimmistä osista viitekehyksessä on palvelurekisteri. Suunniteltaessa *bundle*'ia on oltava huolellinen siitä, ettei käytä sellaisia palveluobjekteja, jotka on poistettu viitekehyksestä. Palvelurekisterin dynaaminen luonne aiheuttaa tarpeen seurata mitä palveluobjekteja viitekehykseen asennetaan ja siitä poistetaan. Tätä tarvetta täyttämään on viitekehyksen määritelmään sisällytetty palvelunseuranta (Service Tracker) –toiminto.

Määritelmässä on erityinen apuohjelmaluokka palvelunseuranta, mikä tekee palveluiden asentamisen, poistamisen ja muuttamisen seurannasta paljon yksinkertaisempaa. Palvelunseuranta -olion tehtäviin kuuluu: palveluiden käynnistysluettelon laatiminen palvelun omistajan välittämistä tiedoista, seurata omistajan haluamien palveluiden tilaa tarkkailemalla `ServiceEvent`-instansseja sekä antaa omistajan muokata seurantaprosessia.

Palvelunseuranta –olio liittyy jokaiseen palveluun, mitkä vastaavat sille annettua hakukriteerejä ja jää seuraamaan palveluun liittyviä `ServiceEvent`-tapahtumia. Näiden tapahtumien perusteella palvelunseuraaja välittää muille palveluille tiedon, jos jossakin seurattavissa palveluissa tapahtuu muutos. Tällainen muutos voi tulla kyseeseen esimerkiksi silloin kun jokin vanha palvelu poistetaan viitekehyksestä. Tällöin sekä viitekehyksen että poistettua palvelua käyttävän palvelun tulee sopeutua uuteen tilanteeseen. Viitekehys hakee poistetun palvelun tilalle uuden palvelun, jos sellainen löytyy. Siinä tapauksessa että korvaavaa palvelua ei löydy, palvelu poistaa sen toiminnon käytöstä, joka käytti poistettua palvelua. [14]

Palvelunseuraaja rakentuu kahdesta osasta (Kuva 22), palvelun seuraajasta sekä seuraajan hallinnallisesta osasta. Seuraajan hallinnallisella osalla määritellään haku kriteerit, joiden mukaan valitaan seurattavat palvelut.



Kuva 22. Palvelun seurantapalvelun rakenne. [14]

2.6.3 Laitteiden haku

Palvelualusta on monien eri laite- ja palvelutoimittajien tuotteiden kohtauspaikka, jossa käyttäjät tilaavat ja poistavat palveluita, uudet asennetut palvelut hakevat sopivat syöttö- ja näyttölaitteet ja laiteohjaimet yhdistävät itsensä oikeisiin laitteisiin. OSGi-ympäristössä edellä mainitut toimenpiteet tapahtuvat viitekehyksen ollessa toiminnassa. Tämä viitekehyksen joustavuus tekee kaikkien mahdollisten OSGi-ympäristön kokoonpanojen määrittelyn vaikeaksi, etenkin laitteista riippuvien kokoonpanojen. Kun kaikki asiakkaan haluamat palvelut ja laitteet ovat asennettuna palvelualustaan, muodostuu OSGi-ympäristöstä ainutlaatuinen. Tämän takia automatisoitu mekanismi on havaittu tarpeelliseksi mukauttamaan OSGi-ympäristö vastaamaan asiakkaan vaatimuksia. Samalla voidaan minimoida tarvetta konfiguroida OSGi-ympäristöä.

Viitekehysmääritelmään on tämän vuoksi lisätty laiteelle pääsy (Device Access) -määritelmä, jolla sovitaan kuinka laitteiden automaattista havaitsemista ja loogista liittämistä OSGi-ympäristössä voidaan tukea ja koordinoida. Lisäksi määrittelyllä pyritään helpottamaan uusien laitteiden liittämistä käynnissä olevaan OSGi-ympäristöön sekä laitteiden poistamista siitä. Samalla tuetaan laiteajureiden lataamista ja asentamista.

Määritelmässä ei kuvata mitään tiettyä laite- tai verkkoteknologiaa eikä mitään tiettyä laitteiden havaitsemismenetelmää. Sen sijaan määritelmässä keskitytään tarkastelemaan sitä, kuinka eri laitevalmistajien laitteet voidaan liittää loogisesti palvelualustassa. Määritelmässä korostetaan standardoitujen laiterajapintojen kehityksen rajoittumista laiteluokkiin. Kuitenkaan yhtään laiteluokkaa ei määritelmässä ole määritetty.

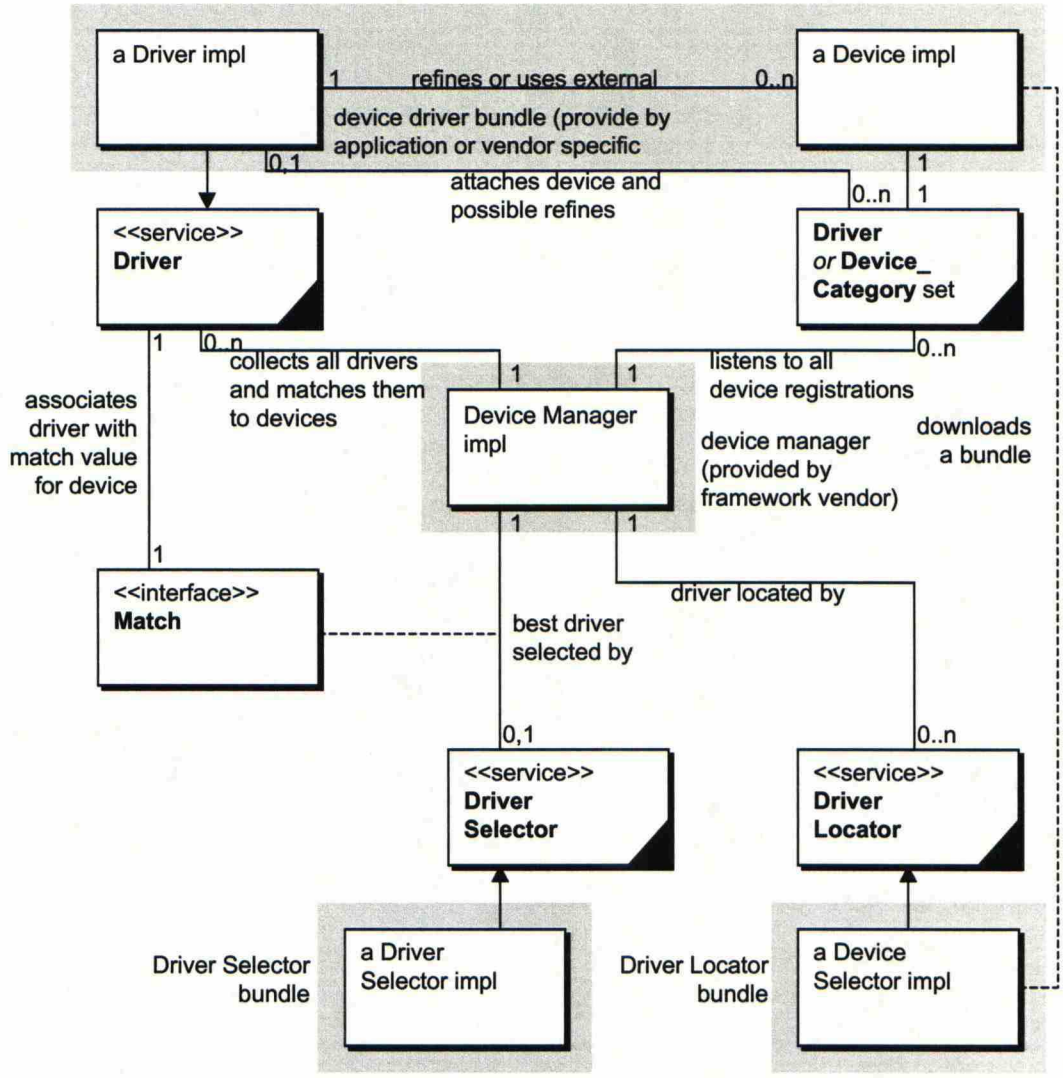
Lähtökohdiksi laitteiden haulle voidaan määritellä muutama kohta. OSGi *bundle*'eita suoritetaan sulautetuissa järjestelmissä. Tämä tarkoittaa usein myös sitä, että käyttäjällä on rajoitetut mahdollisuudet vaikuttaa laitteen toimintaan, sekä sitä että yksinkertaisimmissa laitteissa tulee olemaan selkeitä resurssirajoituksia. Tällaisissa laitteissa tulee olla myös mahdollisuus etähallintaan ja näin ollen tuki etäpalvelun tarjoajan hallintaan.

Jotta palvelualustan avoimuus toteutuu, tulee palveluympäristön olla puolueeton toimittajia kohtaan. Tämän vuoksi eri toimittajien OSGi-yhteensopivien ajuri*bundle*'ien tulee olla hyvin määriteltyjä, dokumentoituja ja korvattavissa olevia. Korvattavuudella tarkoitetaan sitä, että jokainen yksittäinen ajuri tulisi olla korvattavissa ilman, että se vaikuttaa palvelualustan muuhun toimintaan. Esimerkiksi ajurin poistaminen ei saa aiheuttaa viitekehyksen pakollista käynnistämistä uudelleen ennen kuin normaali toiminto voi jatkua. Tätä toimintoa kutsutaan dynaamiseksi päivittämiseksi.

Edellisten vaatimusten lisäksi tulee ajureiden ja palveluidenkin olla vakaita toiminnaltaan ja resurssien käytöltään. Nämä lisävaateet varmistavat palveluympäristön jatkuvan toiminnan ilman uudelleen käynnistyksiä.

2.6.3.1 Laitteiden hakupalvelun rakenne

Laitteiden haku koostuu kuudesta erillisestä kokonaisuudesta (Kuva 23): laitehallintaohjelmasta (device manager), laiteluokista (device category), ajureista, laitteista, ajureiden paikallistajista (driver locator) ja ajurin valitsijasta (driver selector). Laitehallintaohjelma, kuvan keskellä, on *bundle*, joka hallitsee taustalla laitteiden kiinnittämisprosessin aloittamista. Laiteluokka vastaavasti määrittelee kuinka laitepalvelu ja ajuripalvelu voivat tehdä yhteistyötä.



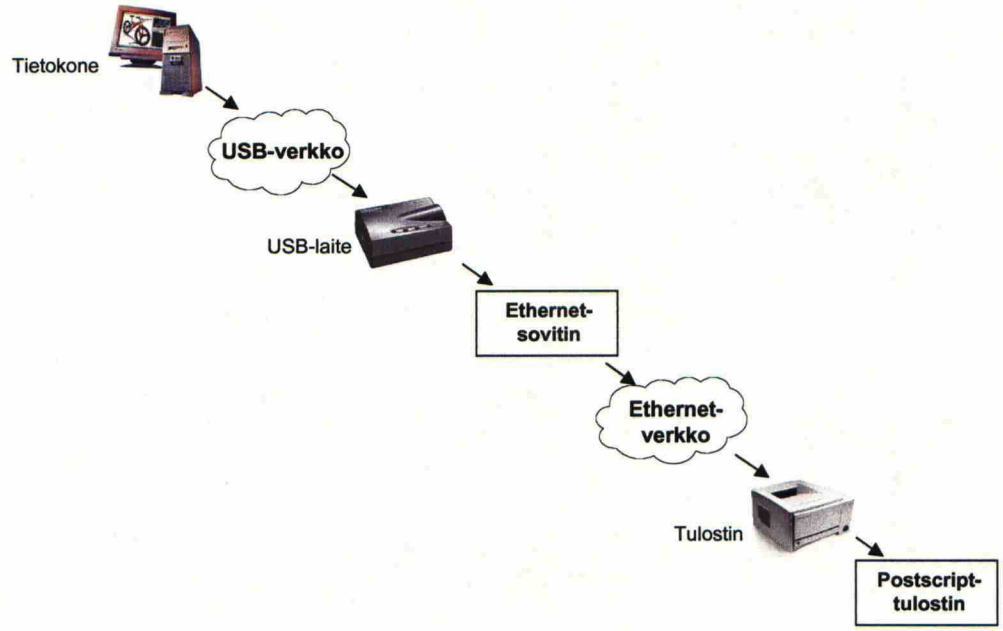
Kuva 23. Laitteiden hakupalvelun rakenne. [15]

Ajurit kilpailevat keskenään siitä, mikä ajureista valitaan liittämään laite sopivaan laiteluokkaan. Laite on kuvaus fyysisestä laitteesta, joka voidaan liittää käyttäen ajuripalvelua. Ajurin paikallistaja on toiminto, joka auttaa ajuripalvelun tarjoavien *bundle*'ien löytämisessä. Vastaavasti ajurin valitsintoiminnon tehtävänä on valita parhaiten laitteelle sopiva ajuri. [15]

2.6.3.2 Laitepalvelu

Laitepalvelu kuvaa jotain fyysistä laitetta. Laitepalvelut vaihtelevat paljon jotkut kuvaavat yksittäisiä fyysisiä laitteita, kun toiset kuvaavat kokonaisia verkkoja. Sama laitepalvelu voi jopa esittää saman laitteen samanaikaisesti eri abstrakteilla tasoilla. Esimerkkinä voidaan mainita tilanne, missä tietokone on liitetty Ethernet verkkoon USB-sovittimella. Samaan Ethernet verkkoon on myös liitetty käyttäjän tarvitsema

PostScript-tulostin. Tällöin tietokone näkee useita eri laitteita riippuen tarkasteltavasta tasosta: USB-verkko, USB verkkoon liitetty laite, sama laite tunnistetaan USB Ethernet sovittimeksi, tunnistetaan laite Ethernet-verkosta, tunnistetaan Ethernetin laite tavalliseksi tulostimeksi ja tarkennetaan tulostin PostScript (PS) tulostimeksi (Kuva 24). Voidaan sanoa, että sama palvelualustaan liitetty laite on laitteen kannalta kaikkia edellä mainittuja aina USB-verkosta PS-tulostimeen. [15]



Kuva 24. Tietokone näkee useita eri laitteita riippuen tarkasteltavasta tasosta.

Toisaalta laite voidaan kuvata myös eri tavoin. Otetaan esimerkiksi USB-hiiri. Hiirtä voidaan alun pitäen pitää USB-laitteena, joka toimittaa tietoa USB-väylää pitkin. Toisaalta se on myös laite, joka toimittaa x- ja y-koordinaatit sekä tietoa sen näppäinten tilasta. Kummallakin kuvauksella on omat vaikutuksensa. Tiedolla siitä, että tämä tietty laite on hiiri ei ole merkitystä sovellukselle, joka tarjoaa USB-rajapinnan hallinnan. Toisaalta se, että hiiri on kytketty USB-väylään tai sarjaporttiin on merkityksetöntä sovellukselle, joka vastaa hiiren liikkeiden vastaanottamisesta.

Laitepalvelun täytyy kuulua johonkin laiteluokkaan tai muuten sen tulee toteuttaa yleinen palvelu, joka kuvaa tietyn laitteen riippumatta laitteen valmistajasta tai liitäntätavasta.

Laiteluokka määrittää menetelmät, joita käyttämällä voidaan keskustella laitteiden kanssa sekä mahdollistaa yhteensopivuus toisten *bundle*'ien kanssa, jotka perustuvat samaan teknologiaan. Yleiset laitepalvelut sallivat yhteistoiminnan *bundle*'ien välillä, joita ei ole liitetty tiettyyn laiteteknologiaan. Esimerkiksi USB vaatii laiteluokan, jotta olisi mahdollista tehdä USB-väylään liitettyjen laitteiden kanssa keskusteleva ajuri*bundle*.

Todennäköisesti suurin osa laitepalveluista kuvaa jollain tavalla fyysisiä laitteita. Laitepalvelu esitellään kuten muut palvelut OSGi-viitekehyksessä ja niiden hallinta ja toiminnot suoritetaan viitekehyksen palveluiden toimesta.

2.6.3.3 Laitepalvelun liittäminen

Kun laitepalvelu rekisteröidään viitekehykseen on laitehallintaohjelma vastuussa sopivien ajuripalveluiden löytämisestä ja niiden liittämisestä juuri rekisteröityihin laitepalveluihin. Itse laitepalvelu on passiivinen, se ainoastaan rekisteröi laitepalvelun viitekehykseen ja jää sen jälkeen odottamaan kunnes sitä kutsutaan.

Viestien välittämistä fyysisille laitteille ei ole määritelty OSGi-palvelualustamäärityksessä, koska se vaihtelee suuresti erilaisten laitteiden välillä. Ajuripalvelu vastaa laitteen liittämisestä laitetyypin mukaisella tavalla. Säännöt ja rajapinnat tätä prosessia varten täytyy erikseen määritellä sopivassa laiteluokassa.

Jos laitehallintaohjelma ei pysty löytämään sopivaa laitepalvelua, jää laitepalvelu toistaiseksi liittämättä. Laitepalvelu voi odottaa kunnes uusi ajuri asennetaan tai se voidaan poistaa ja asentaa uudelleen käyttäen erilaista esimerkiksi yleisempää kuvausta laitteen ominaisuuksista. Laitepalvelua, jota viitekehyksen mukaan ei käytä yksikään *bundle*, kutsutaan käyttämättömäksi laitepalveluksi (idle Device service).

Kun laitepalvelu poistetaan, ei välittömiä toimenpiteitä vaadita laitehallintaohjelmalta. Normaalit viitekehyksen tarjoamat palveluiden poistotapahtumat välittävät tarvittavan tiedon muutoksista laitepalveluista niitä koskeville ajureille. Ajurin tulee vapauttaa tällaisessa tilanteessa laitepalvelu ja poistaa laitepalveluun liittyvät tiedot viitekehyksestä. Tietojen puhdistaminen varmistaa viitekehyksen muistiresurssien riittävyyden muiden palveluiden käyttöön.

Laitehallintaohjelma voi tulkita laitteen poistamisen myös viitteeksi siitä, että laite*bundle*'it ovat käyttämättöminä ja siten myös poistettavissa. Tämän vuoksi on tärkeää, että laitepalvelut poistavat palveluolion kun alapuolinen laiterajapinta poistuu käytöstä.

2.6.3.4 Laiteluokka

Laiteluokka kuvaa ne säännöt ja rajapinnat, joita tarvitaan laitepalvelun ja ajuripalvelun väliseen keskusteluyhteyteen. Vain samaan laiteluokkaan kuuluvat laite- ja ajuripalvelut voivat keskustella keskenään ja toimia yhdessä.

Laiteluokat liittyvät aina tiettyyn laiteteknologiaan, kuten USB, IEEE1394 (Firewire), JiNi, JXTA (Juxtapose) ja UPnP. Laiteluokka määrityksen tarkoituksena on muodostaa kaikille saman luokan laitteille sovittu rajapinta. Näin esimerkiksi toimittaja A:n USB ajurilla voidaan hallita USB-väylään kytkettyjen toimittaja B:n laitteiden laitepalveluja. Alla on yksinkertainen malli kuvitteellisesta laiteluokasta. Laitteiden tässä laiteluokassa tulee toteuttaa rajapinta `com.acme.widget.WidgetDevice` ja liittyä johonkin tämän luokan ajuripalveluun.

```
public interface com.acme.widget.WidgetDevice {
  int MATCH_SERIAL          = 10;
  int MATCH_VERSION         = 8;
  int MATCH_MODEL           = 6;
  int MATCH_MAKE            = 4;
  int MATCH_CLASS           = 2;
  void sendPacket( byte [] data );
  byte [] receivePacket( long timeout );
}
```

Laite- ja ajuripalvelut liitetään toisiinsa yhteensovitusprosessin avulla. Tästä prosessista kerrotaan tarkemmin hieman myöhemmin kohdassa laitehallintaohjelma. Ajuripalvelu on keskeisessä roolissa valittaessa laitepalvelu-ajuripalvelu pareja. Ajuripalvelu tarkastaa uudet juuri rekisteröidyt laitepalvelut ja päättää voisiko se toimia yhdessä tämän laitepalvelun kanssa. Lopputuloksena yhteensovitusprosessista saadaan arvo, joka kuvaa yhteensopivuuden laatua. Yhteensopivuusasteikko määritellään laiteluokassa. Näin toimimalla tarjotaan kaikille ajureille mahdollisuus tasapuoliseen vertailuun.

Laitepalveluiden tulee palauttaa jokin laiteluokassa määritellyistä yhteensopivuusarvoista tai `Device.MATCH_NONE`, jos yhteensopivuutta ei ole. Laiteluokan tulee määritellä selkeät säännöt, kuinka yhteensopivuusarvot

määritellään. Käyttetyssä esimerkissä arvot vaihtelevat 0 ja 10 välillä. Nämä arvot on kuvattu seuraavassa taulukossa (Taulukko 1) tarkemmin.

Yhteensopivuustaso	Arvo	Kuvaus:
MATCH_SERIAL	10	Täydellinen yhteensopivuus mukaan lukien laitteen sarjanumeron
MATCH_VERSION	8	Yhtenevä luokka, valmistaja, malli ja versio
MATCH_MODEL	6	Yhtenevä luokka, valmistaja ja malli
MATCH_MAKE	4	Yhtenevä luokka ja valmistaja
MATCH_CLASS	2	Yhtenevä luokka
MATCH_NONE	0	Laitepalvelu on toisesta laiteluokasta.

Taulukko 1. Esimerkki laiteluokan yhteensopivuusarvoista.

Ajuripalvelun tulisi käyttää yhteensopivuustason nimeä, kun se päättää kuinka hyvin se sopii yhteen laitepalvelun kanssa. Esimerkiksi jos ajuripalvelulla on yhtenevä sarjanumero, sen tulisi palauttaa arvo MATCH_SERIAL.

2.6.3.5 Ajuripalvelu

Ajuripalvelu on vastuussa sopivaan laitepalveluun liittymisestä laitehallintaohjelman valvonnassa. Ennen kuin ajuripalvelu voi liittyä laitepalveluun, se kilpailee muiden ajuripalveluiden kanssa laitteen hallinnasta. Jos ajuripalvelu voittaa kilpailun sen tulee liittyä laitepalveluun laiteluokassa määritellyllä tavalla. Tämän jälkeen ajuripalvelu voi täyttää sille annetut tehtävät.

Yksi ajuripalvelu voi rekisteröidä yhden tai useamman toisen laiteluokan laitepalvelun tai se voi tarjota yleisen ajuripalvelun, mikä kuvaa joukon samanlaisia laitteita esimerkiksi tulostimia.

Ajuripalvelu on toteutettu *bundle*'eilla, kuten kaikki muutkin palvelut. Laitehallintaohjelma tunnistaa sen ajuri*bundle*'in viitekehykseen rekisteröimistä ajuripalveluolioista. *Bundle*'ia, joka sisältää yhden tai useamman ajuripalvelun kutsutaan ajuri*bundle*'iksi.

Ajurit luokitellaan kahdeksaan eri ryhmään. Jokainen laiteajuri voi kuulua yhteen taulukossa 2 kuvattuun ryhmään. Taulukossa oleva lista ei ole absoluuttinen, koska ajuripalvelun ei tarvitse sopia mihinkään luettelossa mainituista ryhmistä. Luokittelun tarkoituksena on eritellä ajureiden eri topologiat.

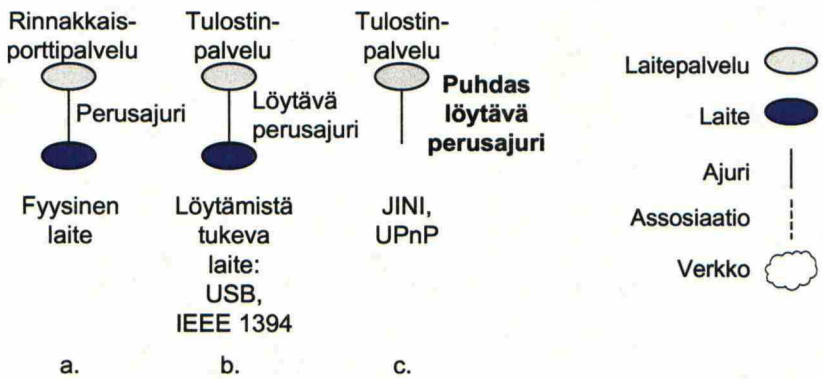
- Perusajuri
- Tarkentava ajuri
- Verkkoajuri
- Komposiittiajuri
- Viittaava ajuri
- Siltaava ajuri
- Yhdistävä ajuri
- Pelkästään käyttävä ajuri

Taulukko 2. Luettelo eri ajuriluokista.

Seuraavassa esitellään tärkeimmät ajuriluokista lyhyesti pienien esimerkkien avulla.

Perusajuri

Ensimmäistä ajuriluokkaa kutsutaan perusajureiksi, koska ne tarjoavat alimman tason kuvauksen fyysisestä laitteesta. Kuvaavin tekijä perusajureissa on se, ettei niitä rekisteröidä ajuripalveluina. Rekisteröintiä ei suoriteta, koska perusajureiden ei tarvitse kilpailla pääsystä niiden alla oleviin teknologioihin.



Kuva 25. Erilaisia perusajureita sekä merkkien selitykset.

Perusajurit löytävät fyysisen laitteen käyttämällä sovittua koodia, esimerkiksi lähettämällä tietyn heksadesimaalisen arvon laitteelle, joka palauttaa arvon muutettuna, ja rekisteröi laitetta vastaavan laitepalvelun (Kuva 25a.) Kun laite tukee löytämismekanismia (discovery mechanism) ja raportoi fyysisestä laitteesta, niin laitepalvelu rekisteröidään. Ajuria, joka tukee löytämismekanismia kutsutaan löytäväksi perusajuriksi (discovery base driver) (Kuva 25b. ja c.). [15]

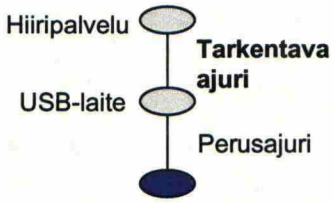
Kaikki teknologiat eivät kuitenkaan tue löytymismekanismia, esimerkiksi suuri osa sarjaporttiin liitettävistä laitteista. Usein on jopa mahdotonta havaita onko laite kytkettynä sarjaporttiin vai ei.

Vaikka jokainen ajuri*bundle* pystyisi havaitsemaan sille kuuluvat laitteet itsenäisesti, tarvitsee löytymistä tukemattoman sarjaportin ajuri ulkopuolista apua. Tämä apu voidaan antaa käyttäjärajapinnan kautta tai laitteen konfiguraatio määritellään konfiguraation hallintaohjelmistolla.

Tarkentava ajuri

Toisen ajuriluokan muodostavat tarkentavat ajurit, jotka tarjoavat tarkemman kuvan fyysisestä laitteesta kuin viitekehykseen jo asennetut laitepalvelut. Tarkennettu ajuri asentaa ajuripalvelun viitekehykseen. Tätä laitepalvelua laitehallintaohjelma käyttää tarkentavien ajureiden liittämisessä epätarkkoihin laitepalveluihin, jotka ovat asennettu viitekehyksen toiminnan yhteydessä. Suurin osa ajureista kuuluu tarkentavien ajureiden ryhmään.

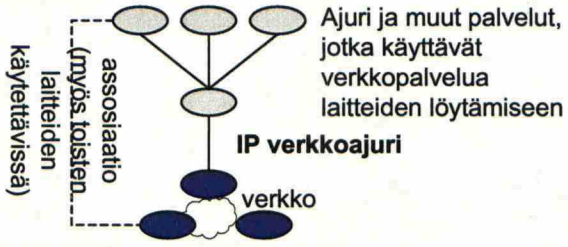
Esimerkiksi tarkennetusta ajurista voidaan kuvata hiiren ajuri, joka on liitetty yleiseen fyysistä hiirtä kuvaavaan USB-laitepalveluun. Hiiriajuri rekisteröi nyt uuden laitepalvelun, joka kuvaa itsensä hiiripalveluksi (Kuva 26). [15]



Kuva 26. Tarkennettu ajuri kuvaa hiiripalvelun.

Verkkoajuri

Internet-protokollan (IP) kanssa yhteensopiva verkko, kuten Ethernet, tukee yksilöllisesti osoitettavia laitteita ja levitystä (broadcasting), mutta siihen ei kuitenkaan ole määritelty varsinaista havaitsemisprotokollaa (discovery protocol). Tässä tapauksessa koko verkko esitetään yhtenä laitepalveluna (Kuva 27).

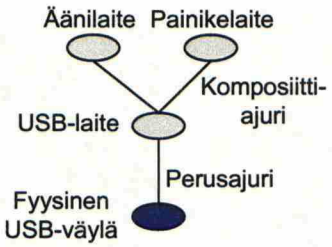


Kuva 27. Verkkoajuri

Komposiittiajuri

Monimutkaiset laitteet voivat usein olla jaettu loogisesti osiin. Ajureita jotka liittyvät yhteen tällaiseen palveluun ja sen jälkeen rekisteröivät useita laitepalveluita kutsutaan komposiitti-ajureiksi. Tällä tavoin voidaan selkeästi vähentää tarvittavien rajapintojen määrää ja lisätä rajapintojen uudelleen käyttöä.

Hyvä esimerkki komposiittiajurista on kuvassa 28, missä USB-kaiutin sisältää muutaman ohjelmistolla ohjattavan painikkeen. Laitteen ajuri voi rekisteröidä nämä painikkeet ja itse laitteen kahdeksi erilliseksi laitepalveluksi, äänipalveluksi ja painikepalveluksi.



Kuva 28. Esimerkki USB-väylään kytketyn kaiuttimen komposiittiajurista.

Viittaava ajuri

Viittaava ajuri ei oikeastaan ole ajuri siinä mielessä, että se hallitsisi jotain laitepalvelua. Sen sijaan viittaava ajuri toimii eräänlaisena välikätenä auttaen löytämään oikean ajuri*bundle*'in. Ajurin hakemisesta kerrotaan tarkemmin kohdassa laitehallintaohjelma.

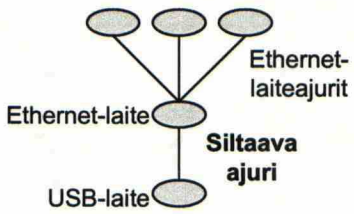
Viittaava ajuri kutsuu liittämismetodia, jotta se selvittäisi laitepalvelulle sopivimman ajuri*bundle*'in. Tämä prosessi voi vaatia yhteyden muodostamisen fyysiselle

laitteelle ja tiedon siirtämistä laiteen ja viitekehyksen välillä. Liittämismetodi palauttaa arvon, joka viittaa ajuri*bundle*'in tunnisteeseen (DRIVER_ID).

Esimerkiksi kuvitteellinen yritys Tacomen voi toteuttaa yhden ajuri*bundle*'in, joka on erikoistunut kaikkien Tacomenin tuottamien laitteiden tunnistamiseen. Viittaava ajuri ei sisällä ohjelmistoa itse laitteen ohjaamiseen, vaan pelkästään logiikan eri laitteiden tunnistamiseen. Tämän vuoksi viittaavan ajurin koko on varsin pieni, vaikkakin se pystyy tunnistamaan varsin laajan tuotelinjan tuotteet. Viittaavan ajurin avulla voidaan vähentää tuntuvasti ajureiden lataamista verkosta sekä niiden vertailua oikean ajuri*bundle*'in löytämiseksi.

Siltaava ajuri

Siltaava ajuri rekisteröi laitepalvelun yhdestä laiteluokasta , mutta liittää sen toisen laiteluokan laitepalveluun. Esimerkiksi tällaisesta ajurista voi mainita USB-väylään kytketyn Ethernet-sovittimen ajurin. Tässä tapauksessa USB-puolen laitepalvelu pinon päällimmäisenä kerroksena on Ethernet-laitepalvelu. Toisaalta sama Ethernet-laitepalvelu voi olla Ethernet-puolen laitepalvelupinon alimmainen kerros. Laitepalveluiden pinoamissyvyyttä ei ole määritelty ja sama ajuri voi esiintyä eri tasoilla saman laitepalvelun pinossa. Kuvassa 29 on kuvattu ajuri- ja laitepalveluiden välisiä kytkentöjä.

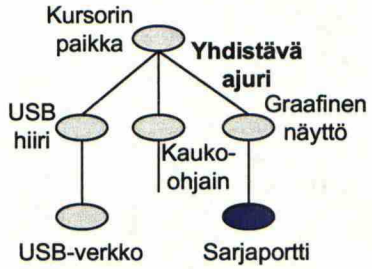


Kuva 29. Siltaavan ajurin rakenne.

Yhdistävä ajuri

Yhdistävä ajuri liittyy useisiin laitepalveluihin ja kerää ne uudeksi laitepalveluksi. Esimerkiksi käy laitteen USB-väylään liitetty tavallinen hiiri, sarjaporttiin piirustusalusta ja kauko-ohjaimen vastaanotin (Kuva 30). Jokainen edellä mainituista laitteista asennetaan palveluina viitekehykseen. Yhdistävä ajuri liittyy näihin

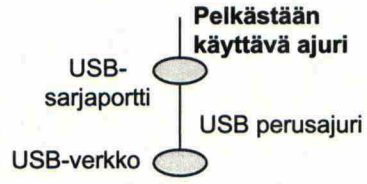
kolmeen palveluun ja yhdistää nämä kolme osoitinohjauslaitetta yhdeksi yleiseksi kursorin paikan määrittelevaksi palveluksi. [15]



Kuva 30. Yhdistävän ajurin rakenne osoittimen paikan määrittelevässä palvelussa.

Puhtaasti käyttävä ajuri

Puhtaasti käyttävä ajuri liittyy laitteeseen ilman, että se rekisteröi tarkennettua ajuria. Esimerkiksi yksi ajuri*bundle* voi päättää käsittelevänsä sarjaportteja Java-suoritusympäristön tarjoaman `javax.comm` avulla, sen sijaan että rekisteröisi sarjaportit palveluiksi. Kun USB-sarjaportti kytketään, yksi tai useampi laitepalvelu liitetään sarjaporttiin. Tämä johtaa siihen, että laitepalvelu pinoutuu sarjaporttilaitepalvelun kanssa (Kuva 31). Käyttävä ajuri voi tällöin liittyä sarjaporttilaitepalveluun ja rekisteröidä uuden sarjaportin `javax.comm.*` rekisteriin viitekehyksen palvelurekisterin sijasta. Tällä tavalla on mahdollista siirtää laite tehokkaasti OSGi-ympäristöstä haluttuun ympäristöön.



Kuva 31. Puhtaasti käyttävän ajurin rakenne.

Ajuripalvelun toteutus

Seuraavassa on yksinkertainen esimerkki ajuripalvelun toteutuksesta. Se muodostuu kahdesta luokasta, joista ensimmäinen on `SerialWidget`-luokka. Tämä luokka tarkkailee yksittäistä `WidgetDevice`-laitetta (kts. vertailulistaus kohdassa laiteluokka) ja rekisteröi `javax.comm.SerialPort`-palvelun (kohta I). Tämä palvelu on yleinen sarjaportti-määritelmä, joka voidaan toteuttaa myös muista laiteluokista kuten USB,

COM-portti jne.. `Javax.comm.SerialPort`-palvelu luodaan, kun laitehallintaohjelma pyytää `SerialWidgetDriver`-oliota liittymään `WidgetDevice`-laitteeseen. `SerialWidgetDriver` rekisteröi uuden `javax.comm.SerialPort`-palvelun (kohta II). Palvelun poisto on toteutettu kohdassa III, missä metodi `removedService` ohitetaan ja `SerialPort` poistetaan samalla, kun `WidgetDevice` poistetaan. [15]

```
package com.acme.widget;
import org.osgi.service.device.*;
import org.osgi.framework.*;
import org.osgi.util.tracker.*;

I   class SerialWidget extends ServiceTracker
      implements javax.comm.SerialPort,
         org.osgi.service.device.Constants {
         ServiceRegistration registration;

      SerialWidget( BundleContext c, ServiceReference r ) {
         super( c, r, null );
         open();
      }

      public Object addingService( ServiceReference ref ) {
         WidgetDevice dev = (WidgetDevice)
            context.getService( ref );
II       registration = context.registerService(
               javax.comm.SerialPort.class.getName(),
               this,
               null
            );

         return(dev);
      }

      public void removedService( ServiceReference ref,
         Object service
      ) {
III      registration.unregister();
         context.ungetService(ref);
      }
      ... methods for javax.comm.SerialPort that are
      ... converted to underlying WidgetDevice
   }
```

`SerialWidgedDriver`-olio rekisteröidään viitekehykseen käyttäen *bundle*-aktivoijan `käynnistä`-metodia. Laitehallintaohjelman tulee kutsua kullekin viitekehyksessä rekisteröitynä olevalle käyttämättömälle laitepalvelulle `vertaa`-metodia (kohta IV). Jos laitehallintaohjelma valitsee esimerkissä käytetyn ajurin ohjaamaan laitepalvelua, luodaan ajurista uusi `SerialWidget`-palvelu. Tämä palvelu tarjoaa muiden *bundle*'ien käyttöön sarjaporttitoiminnallisuudet.

```
public class SerialWidgetDriver implements Driver {
     BundleContext context;

     String spec =
        "(&"
```

```
        +" (objectclass=com.acme.widget.WidgetDevice)"
        +" (DEVICE_CATEGORY=WidgetDevice)"
        +" (com.acme.class=Serial)"
        + ")";

        Filter filter;

        SerialWidgetDriver( BundleContext context )
          throws Exception {
          this.context = context;
          filter = context.createFilter(spec);
        }

        public int match( ServiceReference d ) {
           if ( filter.match( d ) )
             return WidgetDevice.MATCH_CLASS
           else
             return Device.MATCH_NONE;
        }

        public synchronized String attach( ServiceReference r ) {
           new SerialWidget( context, r );
        }
    }
```

IV

Ajuripalvelun rekisteröinti

Ajurit tunnistetaan niiden viitekehykseen asentamien ajuripalveluiden perusteella. Jokaisella ajurilla on aikaisemmin mainittu oma yksilöllinen tunniste `DRIVER_ID`. Ajuripalvelun asentamisen yhteydessä laitehallintaohjelma saa tiedon asennetusta ajurista ja sen `DRIVER_ID`-tunnisteen. Laitehallintaohjelma käyttää `DRIVER_ID`–tunnistetta ajurin kutsumiseen ja samalla varmistetaan, ettei samaa ajuria asenneta uudelleen viitekehykseen. Tämän vuoksi `DRIVER_ID`-tunnisteen tulee riippua ainoastaan ajurin toiminnasta ja olla nimetty niin, että nimen alkuosan muodostaa ajurin toteuttaneen yrityksen käänteinen verkkoaluenimi esimerkiksi `com.wipesec.gadget.1.1`.

Kun ajuripalvelu on rekisteröity, tulee laitteiden liittämisalgoritmia (Device Attachment Algorithm) käyttää jokaiseen käyttämättömään (idle) laitepalveluun. Näin annetaan uusille ajureille mahdollisuus kilpailla laitteista viitekehykseen jo asennettujen ajureiden kanssa käyttämättömistä laitteista. On myös mahdollista, että ajuripalveluolio vastaanottaa `vertaa` (`match`) ja `liitä` (`attach`) pyyntöjä ennen kuin palvelun rekisteröinti-metodi on palautunut.

Ajuripalvelun poistaminen

Kun laitepalvelu on liitetty ajuripalvelulla, voi laitepalvelun vapauttaa ainoastaan sen liittänyt ajuripalvelu. Vastaavasti kun ajuripalvelu poistetaan, ajuripalvelu

vapauttaa kaikki siihen liitetyt laitepalvelut ja vapautetut laitepalvelut siirtyvät tilaan käyttämätön. Laitehallintaohjelma kerää kaikki käyttämättömät laitepalvelut ja yrittää liittää niitä uudelleen viitekehykseen rekisteröityihin ajuripalveluihin.

Laitehallintaohjelman asentamat ajuripalvelut pysyvät rekisteröityinä niin kauan kun ajuri*bundle* on aktiivinen. Tämän vuoksi ajuripalvelu tulee poistaa vain jos ajuri*bundle* pysäytetään *bundle*'in poistamista tai päivittämistä varten. Toisin sanoen ajuripalveluita ei saa poistaa muistin tai muiden resurssien käyttöä minimoitaessa.

2.6.3.6 Ajurin paikallistajapalvelu

Ajurin paikallistajapalvelu sisältää tiedon siitä, kuinka tietyn laitepalvelun tarvitsemat ajuri*bundle*'it voidaan hakea. Kun viitekehykseen rekisteröidään uusi laitepalvelu, laitehallintaohjelma asentaa automaattisesti kaikki ajurin paikallistajapalvelun löytämät ajuri*bundle*'it.

Ajurin paikallistamispalvelun tarkoituksena on erottaa mekanismi menettelytavasta. Päätöksen uuden *bundle*'in asentamisesta tekee laitehallintaohjelma (mekanismi). Ajurin paikallistamispalvelu taas päättää mikä *bundle* asennetaan ja mistä se ladataan (menettelytapa). *Bundle*'in asentamisella on monia vaikutuksia järjestelmän turvallisuuteen. Tämän lisäksi asentamisprosessiin vaikuttavat myös verkon rakenne ja muut laiteympäristön yksityiskohdat. Ajurin paikallistamispalvelun käyttäminen mahdollistaa palveluporttioperaattorin valita tarpeitaan vastaava strategia *bundle*'ien toimittamiseen.

Ajurin paikallistamispalvelu käyttää myös hyväkseen ajurin `DRIVER_ID`-tunnistetta asennuskelpoisten ajureiden tunnistamisessa. OSGi-ympäristössä voi olla useita eri laitteisiin erikoistuneita ajurin paikallistamispalveluita asennettuna. Laitehallintaohjelman tulee käyttää kaikkien hakupalveluiden tuloksia hyväkseen valitessaan ajuria

2.6.3.7 Ajurin valintapalvelu

Ajurin valintapalvelun tarkoituksena on valita sopivin ajuripalvelu, kaikkien sopivien ajureiden joukosta. Laitehallintaohjelmassa on oletus algoritmi, jonka mukaan se valitsee parhaiten sopivan ajuripalvelun. Ajurin hakualgoritmista kerrotaan tarkemmin kohdassa laitehallintaohjelma. Kun algoritmi ei ole riittävä ja

vaatii näin ollen operaattorin tekemän mukautuksen, voidaan ajurin valintapalvelun toteuttava *bundle* asentaa viitekehykseen. Palvelua voi käyttää ainoastaan laitehallintaohjelma, kun valitaan laitepalvelulle sopivinta ajuria.

Viitekehykseen voi asentaa ainoastaan yhden ajurin valintapalvelun. Tämä rajoitus perustuu laitehallintaohjelman rakenteeseen, missä määritellään, että se ei voi käyttää kuin yhtä ajurin valintapalvelua. Virheellisessä tilanteessa, jossa viitekehyksessä on rekisteröitynä enemmän kuin yksi ajurin valintapalvelu, määritellään käytettävä palvelu `service.ranking`-arvon mukaan. `Service.ranking` on arvo palvelun ominaisuudet-taulukossa ja sen asettaa rekisteröitävä *bundle*, kun *bundle*'ia asennetaan viitekehykseen.

Ajurin valintapalvelu toteuttaa `ServiceSelector`-rajapinnan. Laitehallintaohjelman tulee käyttää tämän rajapinnan tarjoamaa metodia `select(Service-Refernce, Match[])`. Tämä metodi vastaanottaa laitepalvelun palvelusuosituksen (Service Reference) ja joukon `Match`-olioita. Jokainen `Match`-olio sisältää linkin ajuripalvelun `ServiceReference`-olioon ja aiemmin kutsutun `Driver.match`-metodin tuottaman vertailuarvon. Ajurin valintapalvelu tutkii kaikki `Match`-oliot ja päättää mikä ajuripalveluista parhaiten sopii laitepalvelulle. Parhaiten sopivan ajuripalvelun indeksi palautetaan laitehallintaohjelmalle. Jos yksikään `Match`-olioista ei sovi laitepalvelulle, palautetaan laitehallintaohjelmalle arvo `DriverSelector.SELECT_NONE (-1)` ja laitepalvelu jää edelleen tilaan käyttämätön.

2.6.3.8 Laitehallintaohjelma

Laitteille pääsyä hallitaan taustalla laitehallintaohjelman avulla. Laitehallintaohjelma on vastuussa kaikkien laitepalveluita ja ajuripalveluita koskevien rekisteröinti-, muuntelu- ja poistotehtävien käynnistämisestä. Laitehallintaohjelma käyttää näihin tehtäviin edellä mainittuja ajurin valinta- ja paikallistajapalveluita.

Laitehallintaohjelma havaitsee laitepalvelun rekisteröimisen viitekehykseen ja koordinoi niiden liittymistä sopiviin ajuripalveluihin. Sopivien ajuripalveluiden ei tarvitse olla aktiivisina viitekehyksessä ollakseen mukana valintaprosessissa. Laitehallintaohjelman tulee käyttää ajurin paikallistamispalvelua havaitulle uudelle laitepalvelulle sopivien asentamattomien ajuri*bundle*'ien löytämiseen.

Laitehallintaohjelma asentaa ja käynnistää nämä ajuri*bundle*'it ajuripaikallistamispalvelun avulla. Tämän tuloksena viitekehykseen rekisteröidään yksi tai useampia ajuripalveluita. Kaikki saatavilla olevat ajuripalvelut osallistuvat liittämisprosessiin. Ajuripalvelu voi tutkia laitepalvelua `ServiceRegference`-olion avulla kuinka hyvin ajuripalvelu vastaa laitepalvelua.

Jos viitekehyksen palvelurekisteriin on rekisteröity ajurin valintapalvelu, käytetään sitä parhaiten sopivan ajuripalvelun valitsemiseen. Jos ajurin valintapalvelua ei ole saatavilla, korkeimman tarjouksen tehnyt ajuripalvelu voittaa. Tasatulos-tilanteissa määräävät palvelun ominaisuustaulukossa olevat `service.ranking`- ja `service.id`-arvot. Tämän jälkeen valittua ajuripalvelua pyydetään liittymään laitepalveluun.

Tilanteessa, jossa yhtään sopivaa ajuripalvelua ei löydy laitepalvelulle, jää laitepalvelu tilaan käyttämätön. Kun viitekehykseen asennetaan uusia ajuripalveluita, suoritetaan kaikille uusille ja käyttämättömille laitepalveluille edellä kuvattu ajuripalvelun valintaprosessi uudelleen.

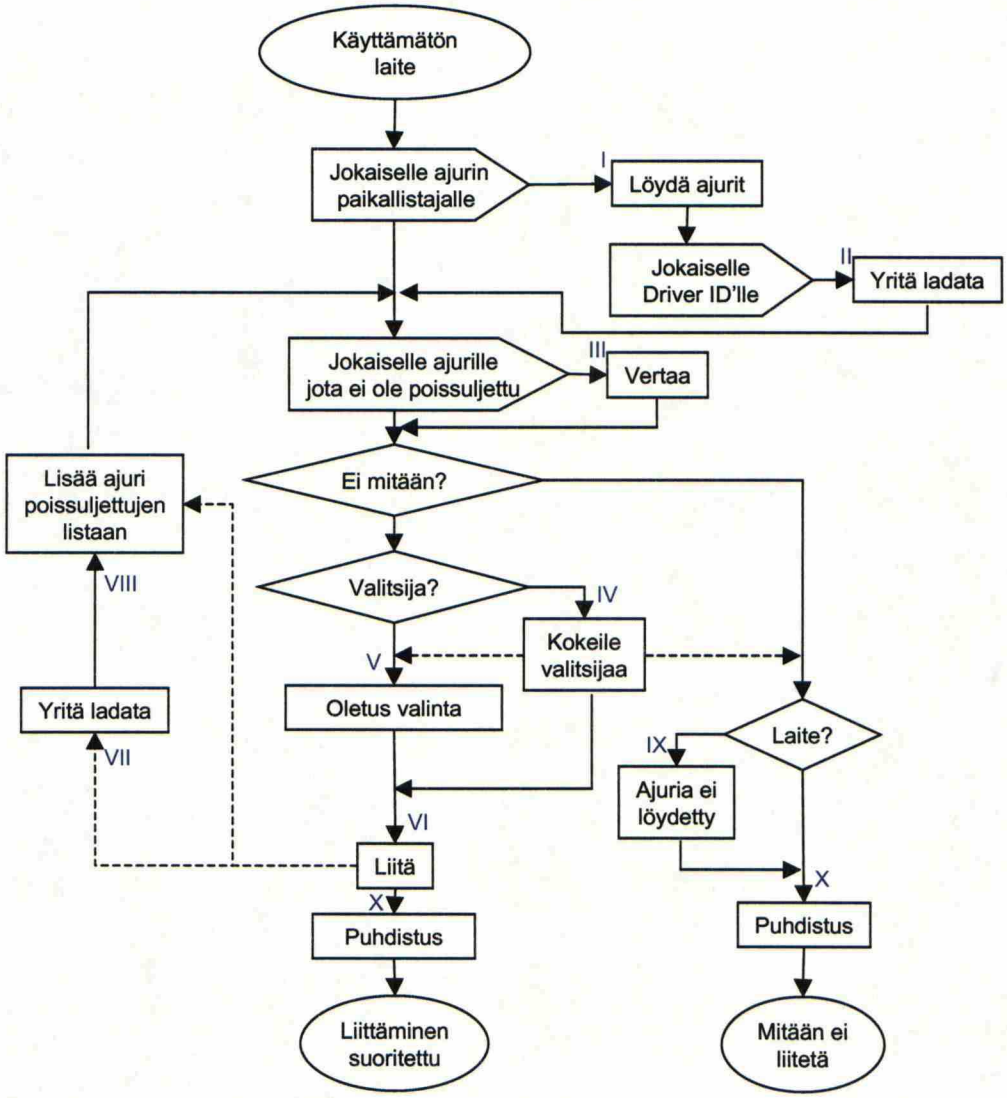
Laitehallintaohjelma liittää laitepalvelun uudelleen, jos jossain tilanteessa myöhemmin laitepalveluun liitetty ajuripalvelu poistetaan tai se päivitetään. Vastaavasti laitehallintaohjelma poistaa sellaiset ajuripalvelut, joita ei enää käytetä viitekehyksessä.

Viitekehykseen voi asentaa ja rekisteröidä ainoastaan yhden laitehallintaohjelman eikä sillä ole ollenkaan julkista rajapintaa, vaan kaikki toiminnot tapahtuvat ajurin paikallistamis- ja valintapalveluiden kautta.

Jotta voidaan välttää ylimääräiset kilpailutilanteet viitekehystä käynnistettäessä, niin laitehallintaohjelma tarkkailee viitekehykseen asennettujen laite- ja ajuripalveluiden tilaa heti käynnistyttyään. Laitehallintaohjelma ei kuitenkaan liitä ajuripalveluita laitepalveluihin ennen kuin viitekehys on käynnistynyt kokonaisuudessaan. Viitekehys ilmoittaa tästä `FrameworkEvent.STARTED`-viestillä. Jos laitehallintaohjelma käynnistetään vasta viitekehyksen käynnistämisen jälkeen, voi laitehallintaohjelma tarkistaa viitekehyksen tilan järjestelmä*bundle*'in tilasta. [15]

2.6.3.9 Laitteen liittämisalgoritmi

Laitehallintaohjelman tärkein tehtävä on liittää tarkentavat ajurit käyttämättömiin laitepalveluihin. Seuraava kaaviokuva (Kuva 32) kuvaa algoritmia, jonka mukaisesti ajuri liitetään laitepalveluun.



Kuva 32. Laite- ja ajuripalveluiden liittämisalgoritmi. [15]

Viitekehykseen on voitu asentaa useita ajurin paikallistajia. Uuden laitepalvelun rekisteröityessä kaikkien ajurin paikallistamispalveluiden `löydä ajuri` –metodia (`DriverLocator.findDrivers`) kutsutaan ja niille toimitetaan tiedot uudesta laitteesta (kohta I). Metodi palauttaa sopivien ajuri*bundle*'ien `DRIVER_ID` arvot.

Laitehallintaohjelma yrittää ladata löytyneistä ajureista ne, joiden `DRIVER_ID` ei vielä ole rekisteröity viitekehykseen (kohta II). Tämä tapahtuu kutsumalla `lataa ajuri` -

metodilla (`DriverLocator.loadDriver`), joka palauttaa varsinaisen ajuri*bundle*'in. Jos `lataa ajuri` –metodi epäonnistuu, prosessia jatketaan toisen ajurin paikallistajan löytämällä oliolla. Jos kaikki tähän `DRIVER_ID`'n liittyvien ajuriolioiden lataus epäonnistuu, ajuri*bundle* hylätään. Jos lataaminen onnistuu, laitehallintaohjelma asentaa ajuri*bundle*'in ja se rekisteröityy viitekehyksen palvelurekisteriin ajuripalveluna.

Kaikkia ajuripalveluita, pois lukien poissuljettujen listalla (exclusion list) olevia ajuripalveluita, verrataan `Driver.match`-metodilla (kohta III). Tällöin `Driver.match`-metodi toimittaa palveluviiteolion laitepalvelulle, joka kerää kaikki hyväksyttävät vertailutulokset. Vertailutulos on hyväksyttävä, jos vertailu arvo on suurempi kuin 0 (Taulukko 1).

Jos viitekehykseen on asennettu ajurin valintapalvelu, kutsuu laitehallintaohjelma `DriverSelector.select`-metodia ja toimittaa sille vertailun tulokset (kohta IV). Ajurin valintapalvelun palauttaessa yhden tunnisteen vertailutuloksista, liitetään tunnistetta vastaava ajuripalvelu laitepalveluun. Vastaavasti jos ajurin valintapalvelu palauttaa arvon `DriverSelector.SELECT_NONE`, ei mitään ajuripalvelua liitetä laitepalveluun ja laitepalvelu jää tilaan käyttämätön.

Jos viitekehykseen ei ole asennettu ajurin valintapalvelua, niin ajuri valitaan suoraan vertailutuloksesta (kohta V). Tällöin se ajuri jolla on suurin vertailuarvo voittaa. Tasapelitilanteessa tarkistetaan millä ajurilla on korkein `service.ranking`-arvo. Jos vieläkään ei ole löytynyt yksittäistä ajuria, niin voittajaksi määräytyy se jolla on pienin `service.id`-arvo.

Ajuripalvelu liitetään laitepalveluun kutsumalla valitun ajuripalvelun `attach`-metodia. Jos metodi palauttaa tyhjän, on ajuripalvelun liittäminen onnistunut (kohta VI). Muussa tapauksessa metodi palauttaa arvon. Tämä arvo viittaa toiseen ajuripalveluun ja prosessia jatketaan sen perusteella. Jos ajuripalvelun liittäminen epäonnistuu, siirtyy algoritmi askeleen eteenpäin kohtaan X.

Laitehallintaohjelma yrittää ladata viittaavan ajurin palauttamaa ajuripalvelua (kohta VII) samoin kuin kohdassa II sillä poikkeuksella, että nyt ei ole tiedossa mitä ajurin paikallistamispalvelua käytetään. Tämän vuoksi käytetään jokaisen ajurin

paikallistamispalvelun `loadDriver`-metodia kunnes yksi paikallistamispalveluista onnistuu ladata ajurin tai kaikki yritykset epäonnistuvat. Jos vain yksi paikantamispalvelu onnistuu, laitehallintaohjelma asentaa ja käynnistää ajuri*bundle*'in. Ajuri*bundle* rekisteröi ajuripalvelun käynnistyessään, mikä lisätään algoritmin ajuripalvelulistaan.

Viittaava ajuri lisätään poissuljettujen listaan (kohta VIII). Koska jokainen uusi viittaus lisää maininnan poissuljettujen listaan, mikä vastaavasti poistaa ajurin vertailulistalta, ei algoritmi voi pyöriä ikuisesti. Poissuljettujen listaa ylläpidetään vain algoritmin suorituksen ajan. Kun algoritmi myöhemmin käynnistyy uudelleen, on poissuljettujen lista jälleen tyhjä.

Jos laitepalveluun ei liitetä yhtään ajuripalvelua (kohta IX), tarkastetaan toteuttaako laitepalvelu `Device`-rajapinnan. Jos laitepalvelu sisältää `Device`-rajapinnan, niin `noDriverFound`-metodia kutsutaan. Tämä voi aiheuttaa laitepalvelun poistamisen ja yhden tai useamman uuden laitepalvelun rekisteröitymisen sen tilalle. Jokainen uusi laitepalvelu aloittaa laitteiden liittämisalgoritmin uudelleen.

Ajuripalvelun ja laitepalvelun liittämisen onnistumisesta tai epäonnistumisesta riippumatta algoritmi on asentanut useita ajuri*bundle*'eja viitekehykseen. Liittämisprosessin lopussa laitehallintaohjelma poistaa kaikki käyttämättömät ajuri*bundle*'it ja näin vapauttaa resursseja tulevaan käyttöön.

Laitehallintaohjelma voi muutenkin optimoida resurssien käyttöä. Se voi muun muassa viivästyttää ajuri*bundle*'in lataamista kunnes ajuriin liitettävää laitetta käytetään ensimmäisen kerran.

Laitteiden hakupalvelu muodostaa yhden viitekehyksen suurimmista turvariskeistä. Laitehallintaohjelma on ainoa *bundle* laitehallintaympäristössä, jolla on oikeus asentaa ja poistaa ajuri*bundle*'eja viitekehyksestä. Laitehallintaohjelman oikeutta asentaa ja poistaa ajureita voidaan käyttää hyväksi vihamielisen tahon hyökkäyksessä. Tämä voisi tapahtua vaikka tarjoamalla viitekehykseen oma ajurin paikallistamispalvelu. Koska ajurin paikallistamispalvelu voi ladata minkä tahansa *bundle*'in ja tämä *bundle* annetaan etuoikeutetun laitehallintaohjelman suoritettavaksi, jolloin vihamielisen tahon tuottama *bundle* voi toteuttaa Troijan

hevosen OSGi-ympäristöön. Tämän vuoksi ajurin paikallistajapalvelun *bundle*'it tarvitsevat erityisen `ServicePermission[REGISTER]`-oikeuden rekisteröidäkseen ajurin paikallistamispalvelun. Palvelualustan operaattorin tulee käyttää suurta harkintaa kenelle tämä rekisteröintioikeus annetaan.

2.6.4 Käyttäjien hallinta

OSGi-ympäristöä käytetään usein paikoissa, joissa käyttäjät tai laitteet käynnistävät erilaisia tehtäviä. On hyvin ymmärrettävää, että nämä tehtävät aiheuttavat tarpeen tunnistaa niiden käynnistäjä. On olemassa useita erilaisia tapoja tunnistaa käynnistäjä esimerkiksi käyttäjätunnuksen ja salasanan avulla, kertakäyttöisen avaimen avulla, biometrisesti tai sertifikaatin avulla.

Kun tehtävän käynnistäjä (initiator) on tunnistettu, tulee tarkistaa onko sillä oikeus käynnistää pyydetty tehtävä. Tämän oikeuden voi antaa ainoastaan OSGi-ympäristön ylläpitäjä ja vaatii siten myös ylläpitoa.

Viitekehyksessä autentikointia ja auktorisointia varten on tehty erillinen käyttäjien hallintapalvelu. Viitekehykseen asennetut *bundle*'it voivat käyttää hyväkseen käyttäjien hallintapalvelun tarjoamia metodeja tunnistaessaan käyttäjiä ja tarkastaessaan heidän oikeuksiaan. Nämä oikeudet esitetään `Authorization`-oliona. Käyttäjien hallintapalvelu tarjoaa käyttäjän tai yleisemmin tehtävän käynnistäjän tunnistamisen Java-kielen oikeusmallin sijasta ohjelman suorittajan perusteella.

Palvelualustan operaattori käyttää käyttäjien hallintapalvelua määrittämään OSGi-ympäristön käyttäjät ja antamaan niille ominaisuudet (properties), valtuutustiedot (credentials) ja roolit (role).

`Role`-olio kuvaa pyynnön antajaa (ihminen, prosessi, ohjelma tms.). Käyttäjien hallintapalveluun on määritelty kaksi roolia käyttäjä (user) ja ryhmä (group). `User`-oliolle voidaan antaa valtuutustietoja, kuten salasana, ja ominaisuuksia, kuten osoite ja puhelinnumero. `Group`-olio on yhdistelmä perus- (basic) ja vaadituista (required) rooleista. Ryhmän rooleja käytetään kun tarkastellaan auktorisointia esimerkiksi tehtävän suorittamiseen.

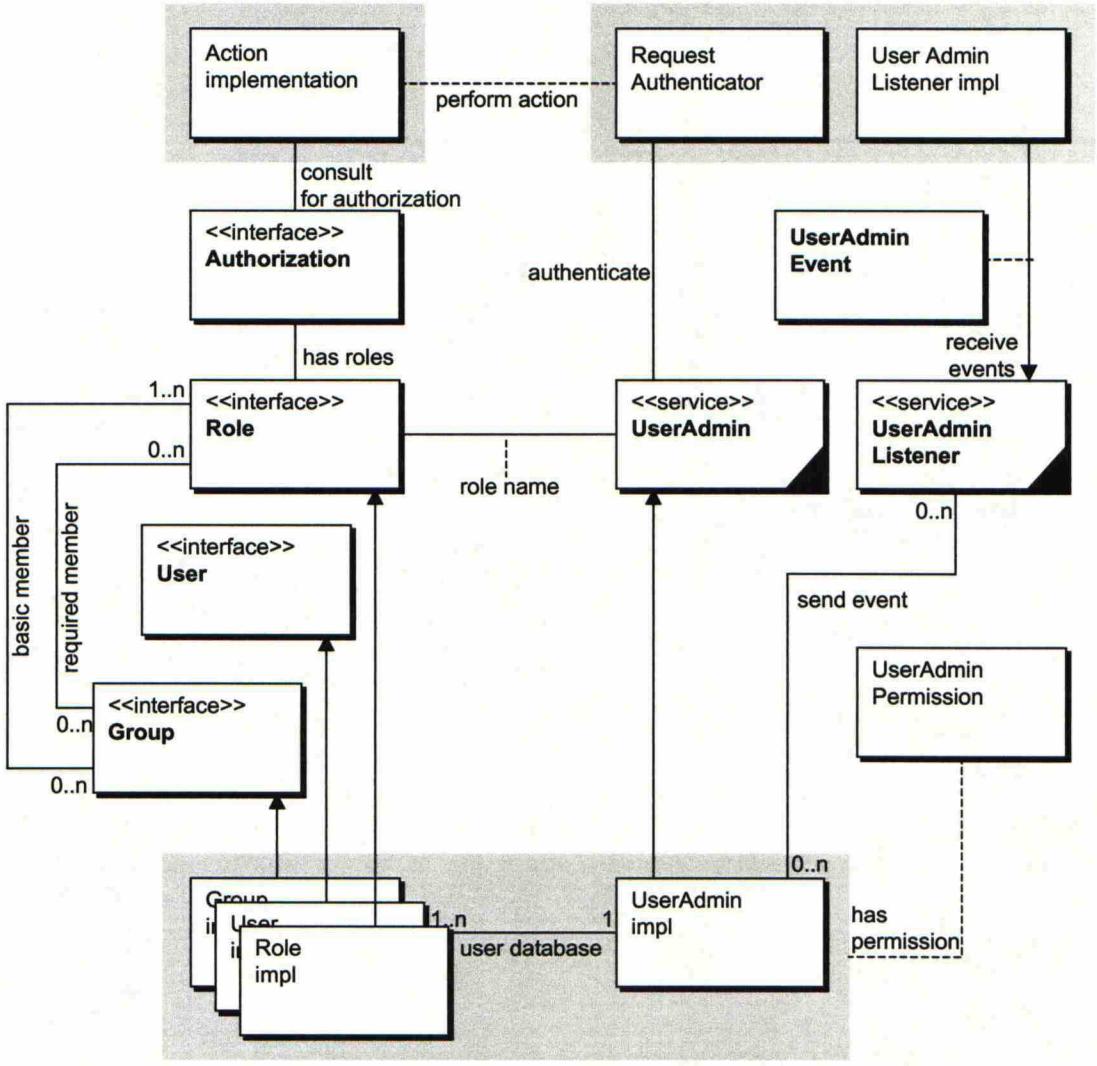
OSGi-ympäristössä voi olla useita ympäristöön tulokohtia. Jokainen näistä kohdista on vastuussa tulevien vaatimusten autentikoinnista. Yhtenä esimerkkinä tulokohdista on http-palvelu (HyperText Transport Protocol), mikä toteuttaa OSGi-ympäristöön oman http-palvelimen. Http-palvelu siirtää tulevien pyyntöjen autentikoinnin http-palvelun rekisteröinnin yhteydessä määritetylle `HttpContext`-olion `handleSecurity`-metodille. OSGi-ympäristön tulokohdat käyttävät käyttäjien hallintapalveluun määriteltyjä tietoja tulevien pyyntöjen autentikointiin. Esimerkki tällaisesta tiedosta on käyttäjän yksityisiin valtuutustietoihin tallennettu salasana tai sertifikaatti.

Bundle selvittää toimintopyynnön auktorisoinnin katsomalla onko olemassa pyydetyn toiminnon kanssa saman nimistä `role`-oliota. Toimintopyyntö voidaan suorittaa, jos toiminnon käynnistäjää kuvaava `role`-olio sisältää pyydettyä toimintoa kuvaavan `role`-olion. Siis käynnistäjän `role`-olio I sisältää toimintoryhmän olion A, jos I sisältää ainakin yhden A:n perusjäsenen sekä kaikki A:n vaaditut jäsenet. Tai I ja A ovat samat. [17]

Edellä mainittu epätriviaali logiikka on toteutettu `Authorization`-luokassa. Käyttäjien hallintapalvelu voi tallentaa autentikoidun `user`-olion erioikeudet `Authorization`-olioon. `Authorization.hasRole`-metodilla tarkastetaan, onko autentikoidulla `user`-oliolla tarvittavaa toimintoryhmän `role`-oliota. Esimerkiksi http-palvelussa `HttpContext`-olio voi autentikoida käynnistäjän ja asettaa `Authorization`-olion pyynnön otsikkoon. Palvelinsovelma (servlet) kutsuu `Authorization`-olion `Authorization.hasRole`-metodia tarkistaakseen, että käynnistäjällä on oikeus suorittaa toiminto.

Käyttäjien hallintapalvelu muodostuu seuraavista kuvassa 33 esitellyistä rajapinnoista ja luokista. `UserAdmin`-rajapinta hallitsee nimettyjen roolien tietokantaa. Tietokantaa käytetään autentikointiin ja auktorisointiin. `Role`-rajapinta esittää roolien ominaispiirteet, kuten nimen, tyypin ja ominaisuuksien luettelon. `User`-rajapinta laajentaa `Role`-rajapintaa ja sitä käytetään esittämään mitä tahansa kokonaisuutta, johon on liitetty valtuutustietoja. Valtuutustietoja käytetään toiminnon käynnistäjän autentikoinnissa. `Group`-rajapinta myöskin laajentaa `Role`-rajapintaa ja sitä käytetään nimettyjen yhdistettyjen `Role`-olioiden, `Group`- tai `User`-olioiden, säilyttämiseen.

Authorization-rajapinta kapseloi auktorisointi kontekstin siitä, mihin *bundle*'iin auktorisointi voi perustua.



Kuva 33. Käyttäjien hallintapalvelun rakenne esitettynä lohkokaaviona. [17]

2.6.4.1 Autentikointi

Autentikoinnilla tarkoitetaan käyttäjän identiteetin varmistamista teknisin keinoin. Laajahko kahtiajako voidaan tehdä nk. heikon tunnistamisen ja vahvan tunnistamisen menetelmien välillä. Heikko tunnistusmenetelmä mahdollistaa identiteetin helpohkon varastamisen esimerkiksi käyttäjätunnuksen ja salasanan lukeminen olan yli. Vahvalla tunnistusmenetelmällä tarkoitetaan sellaista menetelmää, jossa yleensä käytetään jotain matemaattista menetelmää tunnistustapahtuman aikaansaamiseksi. Yleensä vahva tunnistaminen tapahtuu siten, että käyttäjällä on oltava hallussaan jokin fyysinen laite (token) ja sen avaamiseen

käytettävä henkilökohtainen aktivointikoodi. Yleisin tällainen menetelmä on RSA Security Inc.:n SecurID-token, jossa käyttäjällä on vaihtuvan avauskoodin generoiva laite sekä oma henkilökohtainen aktivointikoodi.

Tulevaisuudessa PKI-järjestelmät yhdessä toimikorttien kanssa tulevat luomaan vähintään yhtä vahvan metodin tunnistamiselle, niin kutsutun kryptografiseen haaste-vaste –käsittelyyn perustuvan toiminnon avulla

2.6.4.2 Tietojen säilyttäminen

Käyttäjien hallintapalvelu tarjoaa `Role`-olioille talletuspaikan. Jokaisella `Role`-oliolla on ainutlaatuinen nimi ja kokoelma ominaisuuksia, jotka ovat kaikkien luettavissa sekä muutettavissa kunhan muuttajalla on siihen oikeuttava `UserAdminPermission`. Tarkemmin `UserAdminPermission` oikeusluokasta kerrotaan myöhemmin kohdassa käyttäjien hallintapalvelun turvallisuus. Lisäksi `User`-olioilla on myös kokoelma yksityisiä suojattuja ominaisuuksia, joita kutsutaan valtuutustiedoiksi. Valtuutustiedot ovat ylimääräinen kokoelma ominaisuuksia, joita käytetään käyttäjien autentikoimiseen ja ne on suojattu. Vain tietyt tahot, joilla on riittävät oikeudet, pääsevät valtuutustietoihin käsiksi.

Ominaisuuksiin päästään käsiksi `Role.getProperties()`-metodilla ja valtuutustietoihin vastaavasti `User.getCredentials()`-metodilla. Molemmat metodit palauttavat kirjasto (`Dictionary`)-olion, joka sisältää avain-arvo -pareja. Avaimet riippuvat autentikointimekanismin toteutuksesta, eikä niitä sen vuoksi määritellä OSGi-määrityksessä.

Tietojen tallennuspaikasta voidaan hakea olioita, joilla on ainutlaatuiset ominaisuudet (avain-arvo –parit) `UserAdmin.getUser(String,String)`-metodilla. Metodinkäyttö tekee tiettyyn autentikoimismekanismiin liitetyn käyttäjän löytämisen helpoksi. Esimerkiksi toimikorttimekanismilla, joka generoi ainutlaatuisia avauskoodeja, voi olla sarjanumero, joka identifioi käyttäjän. Tällöin kortin omistaja voidaan löytää alla olevalla metodilla. Jos metodi löytää samat ominaisuudet useilta käyttäjiltä (`User`-oloilta), palauttaa se tyhjän viestin. [17]

```
User owner = useradmin.getUser(
   "secure-card-serial", "132456712-1212").
```

On myös mahdollista hakea tietoa käyttäjän valtuutustiedoista ilman, että itse tietoa valtuutuksista haetaan. Tämä tapahtuu `User.hasCredential(String,object)`-metodia käyttämällä. Valtuutustiedot on suojattu käyttäen `UserAdminPermission`'ia. Koska ominaisuudet ovat kaikkien sellaisten tahojen luettavissa, joilla on pääsy `User`-olioon, `UserAdminPermission` suojaa ainoastaan ominaisuuksien muuttamisen.

Käyttäjien hallintarajapinta on suoraviivainen sovellusliittymä ylläpitämään `User`- ja `Group`-olioiden tallennuspaikkaa. Käyttäjien hallintarajapinta sisältää metodeja, joilla uusia `User`- ja `Group`-olioita voidaan luoda, kuten `createRole(String,int)`-metodi, ja vanhoja `Role`-olioita poistaa. Metodien avulla on myös mahdollista, että samaa allekirjoitusta käytetään tulevaisuudessa uuden tyyppisten `Role`-olioiden luontiin.

Olemassa oleva ympäristöä voidaan tarkastella sellaisten metodien avulla, jotka luetteloivat kaikki ympäristössä olevat `Role`-oliot. Metodi käyttää hyväkseen viitekehyksen yhteydessä esitellyn viitekehyssuotimen kaltaista suodinta, joka palauttaa ainoastaan sellaiset `Role`-oliot, joiden ominaisuudet vastaavat suotimen asetuksia.

2.6.4.3 Tunnistaminen

Autentikointimekanismit, jotka vahvistavat käyttäjän, ovat hyvin yleisiä. Seuraava esimerkissä autentikointi perustuu käyttäjän syöttämään salasanaan.

```
public User authenticate(
     UserAdmin ua, String name, String pwd
) throws SecurityException{
  User user = ua.getUser("com.acme.basicid",
    username);
  if (user == null)
    throw new SecurityException( "No such user" );

  if (!user.hasCredential("com.acme.password", pwd)
    throw new SecurityException(
      "Invalid password" );
  return user;
}
```

Autentikointi-*bundle*'in toimittaja käyttää ominaisuutta `com.acme.basicid` käyttäjän nimen taltiointiin, kun käyttäjä yrittää kirjoittautua sisään OSGi-ympäristöön. Ominaisuutta käytetään käyttäjän `User`-olion paikallistamiseen käyttäjien hallintapalvelun sisällä. `Com.acme.password` –valtuutustieto sisältää nimensä mukaisesti käyttäjän salasanan ja sitä verrataan käyttäjän kirjautumishetkellä antamaan salasanaan. Jos salasana on oikein, palautetaan `User`-olio. Muissa

tapauksissa palautetaan poikkeus `SecurityException` eikä käyttäjää hyväksytä OSGi-ympäristöön.

Tunnistaminen voidaan suorittaa myös käyttäen hyväksi sertifikaatteja. Tällöin järjestelmällä ja käyttäjällä ei tarvitse olla niin sanottua jaettua salaisuutta kuten salasanaa. Tämän sijasta sertifikaatti sisältää nimen, julkisen avaimen ja yhden tai useamman sertifikaatin antajan allekirjoituksen.

Sertifikaatissa olevaa nimeä voidaan käyttää avuksi kun paikallistetaan `User`-oliota käyttäjien hallintapalvelussa, kuten salasanatapauksessa. `User`-olion löytyminen tunnistaa toiminnon käynnistäjän, muttei kuitenkaan autentikoi sitä.

Ensimmäinen askel käynnistäjän autentikoinnissa on tarkistaa, että käynnistäjällä on sertifikaattia vastaava yksityinen avain. Tämän jälkeen käyttäjien hallintapalvelu tarkistaa, että sillä on `User`-olio oikeilla ominaisuuksilla esimerkiksi `com.acme."certificate"="Fudd"`. Seuraavaksi tarkistetaan onko sertifikaatin allekirjoittanut luotettu taho. *Bundle* käyttää keskitettyä listaa kaikista luotetuista allekirjoittajista ja näin ollen hyväksyy vain siinä listassa esitettyjen tahojen allekirjoittamat sertifikaatit. Vaihtoehtoisesti *bundle* voi vaatia että sertifikaatti on tallennettu käyttäjien hallintapalveluun yksilöllisen avaimen mukaan. Kummassakin tapauksessa, kun sertifikaatti on tarkistettu ja hyväksytty on siihen liitetty `User`-olio autentikoitu.

2.6.4.4 Auktorisointi

Valtuuttamisella eli auktorisoinnilla tarkoitetaan käyttäjän tunnistamisen jälkeen tapahtuvaa identiteetin ja käyttöoikeustietokannan yhdistämisoperaatiota. Auktorisointijärjestelmä päättelee käyttäjän identiteetin ja käyttöoikeuspolitiikkojen (policy) avulla, mihin resursseihin hänellä on riittävät oikeudet. Käyttöoikeuksien määrittely riippuu monesti järjestelmän valmistajasta, eikä oikeuksien määrittelystä ole näin ollen muodostunut vakiintunutta tapaa.

OSGi-ympäristössä auktorisointi perustuu roolimalliin. Tässä mallissa jokaiseen toimintaan, jonka *bundle* voi suorittaa, on liitetty jokin rooli. Jokainen rooli on käyttäjien hallintapalvelun ryhmäolio. Esimerkiksi jos palvelinsovelmaa käytetään

hälytysjärjestelmän aktivointiin, niin käyttäjien hallintapalvelussa on oltava ryhmä nimeltä `AlarmSystemActivation`.

Operaattori hallitsee auktorisointia määrittelemällä ryhmiin niihin kuuluvat `User`-oliot eli käyttäjät ja toiset ryhmät. Ryhmiä käytetään käyttäjien hallintaan tarvittavan työn minimoimiseksi. On yksinkertaisinta ja nopeinta luoda yksi `Administrator`-ryhmä ja liittää siihen kaikki sellaiset käyttäjät, jotka tarvitsevat järjestelmänvalvojan käyttöoikeuksia. Tämän jälkeen `Administrator`-ryhmä voidaan liittää kaikkiin hallintarooleihin. Vaihtoehtoinen tapa tehdä sama asia on määritellä jokainen käyttäjä erikseen jokaiseen hallintarooliin. Nyt käyttäjän lisääminen ja poistaminen järjestelmänvalvojaksi voidaan tehdä yksinkertaisesti yhteen paikkaan yhdellä toiminnolla.

Päätös käyttäjän auktorisoinnista voidaan tehdä kahdella eri tavalla. Toiminnon käynnistäjällä on oikeus suorittaa toiminto, joka esitetään ryhmänä, jos se kuuluu johonkin ryhmäolion jäsenistä. Esimerkiksi aiemmin mainittu `AlarmSystemActivation`-ryhmäolio sisältää `Administrator`- ja `Family`-ryhmäoliot.

```
Administrators = { Matti, Pirkko, Samuli }
Family = { Matti, Pirkko, Sirpa, Joonas }

AlarmSystemActivation = { Administrators, Family }
```

Nyt kaikki viisi jäsentä Matti, Pirkko, Samuli, Sirpa ja Joonas voivat aktivoida hälytysjärjestelmän.

Vaihtoehtoisesti käynnistäjän voidaan sallia toiminnon suorittaminen, jos käynnistäjä on jäsenenä kaikissa ryhmäolioissa. Tässä tapauksessa yllä olevasta määrityksestä ainoastaan Matti ja Pirkko voivat aktivoida hälytysjärjestelmän, koska Samuli, Sirpa ja Joonas eivät ole molempien ryhmäolioiden jäseniä.

OSGi-ympäristön käyttäjien hallintapalvelu tukee edellä kuvattujen tapojen lisäksi niiden yhdistelmää, jossa määritellään joukko perusjäseniä (basic members) ja joukko vaadittuja jäseniä (required members).

```
Administrators = { Matti, Pirkko, Samuli }
Family = { Matti, Pirkko, Sirpa, Joonas }

AlarmSystemActivation
   required = { Administrators }
   basic = { Family }
```

Tässä tapauksessa perusjäsenet kattavat ne käyttäjät, jotka voivat ottaa yhteyden palvelualustaan ja vaaditut jäsenet pienentävät tätä joukkoa vaatimalla, että käynnistäjä sisältää jokaisen vaaditun jäsenen. `User`-olio sisältää `Group`-olion, jos `User`-olio sisältää kaikki `Group`-olion vaaditut jäsenet ja vähintään yhden `Group`-olion perusjäsenen.

Auktorisoinnin monimutkaisuus on piilotettu `Authorization`-luokkaan. Tavallisesti autentikoija noutaa `Authorization`-olion käyttäjien hallintapalvelulta toimittamalla hallintapalvelulle autentikoidun `User`-olion. Tämä `Authorization`-olio toimitetaan edelleen toiminnon suorittavalle *bundle*'ille. *Bundle* tarkistaa auktorisoinnin `Authorization.hasRole(String)`-metodilla. `Authorization`-olio tarkistaa vuorostaan sisältääkö autentikoitu käyttäjä pyydetyn toiminnon `Role`-olion eli toiminnon nimellä nimetyn `Group`-olion. Alla on yksinkertainen ohjelmaesimerkki asiasta. [17]

```
public void activateAlarm(Authorization auth) {
   if ( auth.hasRole( "AlarmSystemActivation" ) ) {
      // activate the alarm
      ...
   }
   else throw new SecurityException(
      "Not authorized to activate alarm" );
}
```

OSGi-ympäristössä http-palvelu on tärkeä tulokohta. Tämän vuoksi se on ollut keskeisessä asemassa kehiteltäessä käyttäjien hallintapalvelua. Http-palvelu siirtää autentikointivastuun palvelinsovelman rekisteröinin yhteydessä määritetylle `HttpContext`-oliolle. `HttpContext`-olio käyttää käyttäjien hallintapalvelua apuna autentikoidessaan käyttäjiä. `User`-olion löytämisen jälkeen `HttpContext`-olio luo `Authorization`-olion `UserAdmin.getAuthorization(User)`-metodilla. Tämä `Authorization`-olio kopioi itselleen kaikki `User`-oliolle määritellyt roolit.

Http-palvelu käyttää palvelinsovelman sovellusliittymää (API, Application Program Interface) tehdessään pyyntöjä palvelinsovelmalla. Tällä hetkellä käytössä oleva sovellusliittymä ei kuitenkaan tue OSGi-ympäristön `Authorization`-oliota. Tämän vuoksi `Authorization`-olio on lisätty attribuuttina `javax.servlet.ServletRequest`-olioon alla kuvatun kaltaisena.

```
org.osgi.service.http.HttpContext.AUTHORIZATION
= ("org.osgi.service.useradmin.authorization")
```

Seuraavassa esimerkissä on yksinkertainen malli miten eri käyttäjäryhmiä voidaan käyttää hyväksi. Tässä operaattori asentaa OSGi-ympäristön. OSGi-ympäristöön asennetut *bundle*'it voidaan jakaa seuraaviin toimintoryhmiin: hälytysjärjestelmän hallinta, Internet-yhteys, lämpötilan säätö, valokuva-albumin muokkaus, valokuva-albumin esittäminen ja porttien uudelleen ohjaus.

Pois jätetyt *bundle*'ien asentaminen ja poistaminen kuuluvat myös edelliseen toimintolistaan ja edustavat OSGi-ympäristön kannalta kriittisiä toimintoja. Tämän vuoksi operaattori määrittelee myös joukon ryhmiä, jotka sisältävät erityyppisiä järjestelmän käyttäjiä kuten järjestelmänvalvojat, kaverit, lapset, aikuiset ja asukkaat. Alla on esitetty operaattorin kotitalouden käyttäjistä luoma jako.

```
Asukkaat:       Risto, Heidi, Juhani, Taru
Kaverit:        Johannes, Kim, Olavi, Tarja, Pirkko
```

Aluksi asukkaat ja tuttavat on jaettu järjestelmän käyttäjäryhmiin. Tämän jälkeen käyttäjäryhmät jaetaan edelleen toimintoryhmiin. Alla on esitetty käyttäjien oikeuksien jakaminen taulukkomuodossa (taulukko 3 ja 4). Kaikkien OSGi-ympäristön kriittisten toimintojen suoritusoikeus on rajattu vain järjestelmänvalvojille. On kuitenkin mahdollista, että operaattori ei anna käyttäjälle täysiä järjestelmänvalvojan oikeuksia. Esimerkiksi operaattori voi haluta pitää itsellään oikeuden esimerkiksi asentaa ja poistaa *bundle*'eita ja samalla estää käyttäjää asentamasta omia *bundle*'eita.

Ryhmä	Risto	Heidi	Juhani	Taru	Johannes	Kim	Olavi	Tarja	Pirkko
Asukkaat	Perus	Perus	Perus	Perus	-	-	-	-	-
Kaverit	-	-	-	-	Perus	Perus	Perus	Perus	Perus
Lapset	-	-	Perus	Perus	-	-	-	-	-
Aikuiset	Perus	Perus	-	-	-	-	-	-	-
Järjestelmänvalvojat	Perus	-	-	-	-	-	-	-	-

Taulukko 3. Kotitalouteen liittyvien käyttäjien jako eri käyttäjäryhmiin.

Ryhmät	Asukkaat	Kaverit	Lapset	Aikuiset	Järjestelmänvalvojat
Hälytysjärjestelmä	Perus	-	-	-	Vaadittu
Internet yhteys	Perus	Perus	-	Vaadittu	-
Lämpötilan hallinta	Perus	-	-	Vaadittu	-
Valokuva-albumin muokkaus	Perus	-	Perus	Perus	-
Valokuva-albumin esittäminen	Perus	Perus	-	-	-
Porttien uudelleen ohjaus	Perus	-	-	-	Vaadittu

Taulukko 4. Käyttäjäryhmien jako peruskäyttäjiin ja vaadittuihin käyttäjiin.

Taulukosta 3 nähdään, että Risto kuuluu käyttäjäryhmiin asukkaat, aikuiset ja järjestelmänvalvoja, sekä sen että Taru kuuluu ryhmiin asukkaat ja lapset. Kun nämä tiedot siirretään taulukkoon 4 huomataan, että ainakin Riston tai Heidin on oltava kotona, jotta Taru voi selailla www-sivuja Internetistä. Vastaavasti vaaditaan, että Risto on paikalla kun hälytysjärjestelmä aktivoidaan tai sammutetaan.

2.6.4.5 Tapahtumat käyttäjien hallintapalvelussa

Muutokset käyttäjien oikeuksissa ovat käytettävissä tosiaikaisesti, heti niiden hyväksymisen jälkeen. Jokainen käyttäjien hallintapalvelu toteutus lähettää `UserAdminEvent`-olion kaikille viitekehykseen `UserAdminListener`-rajapintaan rekisteröityneille palveluille. Tätä kutsutaan valkoisen taulun (white board) lähestymistavaksi. Seuraava ohjelmalistaus esittää yksinkertaistetusti lähestymistavan toteutusta.

```
class Listener implements UserAdminListener {
   public void roleChanged( UserAdminEvent event ) {
   ...
   }
}
public class MyActivator
   implements BundleActivator {
   public void start( BundleContext conext ) {
      context.registerService( new Listener() );
   }
   public void stop( BundleContext context ) {}
}
```

Tapahtumien kuuntelijaa (listener) ei tarvitse erikseen poistaa. Kun *bundle* pysäytetään, viitekehys automaattisesti poistaa siihen liitetyn kuuntelijan viitekehyksestä. Kuuntelijan rekisteröinnin jälkeen kaikista muutoksista roolien tallennuspaikassa ilmoitetaan `UserAdminListener`-oliolle.

2.6.4.6 Käyttäjien hallintapalvelun turvallisuus

Käyttäjien hallintapalvelu vastaa OSGi-ympäristön yleistä tietoturvamallia, ja täydentää Java-ympäristön omaa tietoturvamallia (The Java Security Architecture for JDK 1.2). Päätös toiminnon suorittamisesta on siten yhdistelmä Java2-oikeudesta, joka perustuu suoritettavaan ohjelmakoodiin, ja käyttäjien hallintapalvelun auktorisointiin, eli tietoon ohjelman suorittajasta.

Kuten jo aiemmin viitattiin käyttäjien hallintapalvelu määrittelee `UserAdminPermission`-luokan, jota käytetään rajoitettaessa *bundle*'ien pääsyä

valtuutustietoihin. Luokka sisältää toiminnot, joilla voidaan muuttaa käyttäjän ominaisuustietoja, valtuutustietoja sekä hakea valtuutustiedot tietyltä käyttäjältä.

Jos oikeuden nimi on "`admin`", sallii se omistajansa hallita käyttäjien hallintapalvelun tietojen talletuspaikkaa. Tällöin oikeuteen ei ole liitetty mitään toimintoa. Muulloin oikeuden nimi vastaa ominaisuuden nimeä. Nimi voi loppua "`.*`"-merkkijonoon ilmaisten jokerimerkkiä. Esimerkiksi `com.acme.*` vastaa nimiä `com.acme.risto`, `com.acme.heidi` ja niin edelleen.

2.6.4.7 JAAS ja OSGi käyttäjien hallintapalvelu

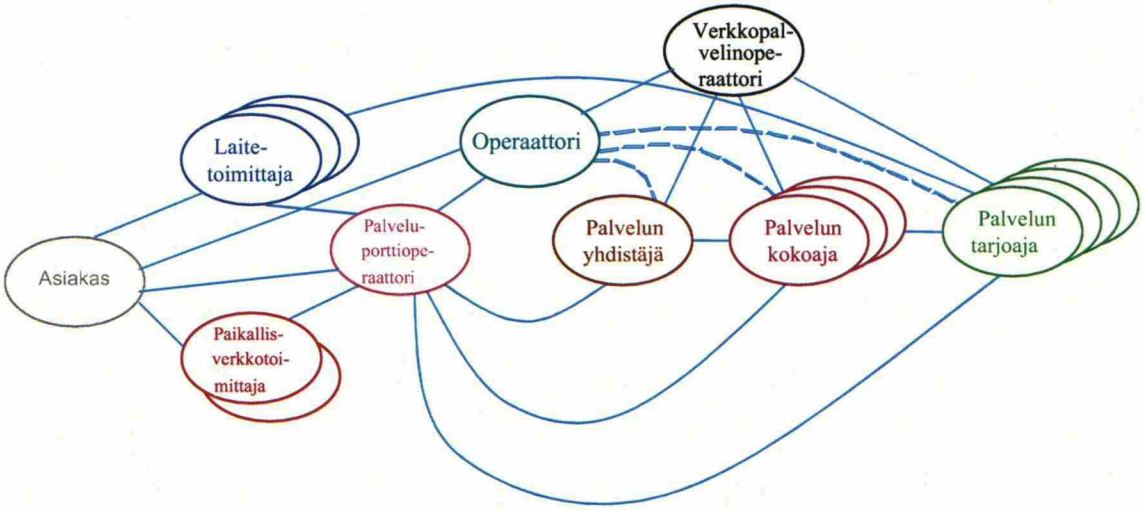
Javan auktorisointi ja autentikointipalvelu (Java Authorization and Authentitication Service, JAAS) näyttää aluksi soveltuvan hyvin käyttäjien hallintaan. OSGi kuitenkin päätti kehittää riippumattoman käyttäjien hallintapalvelun, koska JAAS'ia ei pidetty sopivana. Syynä tähän oli riippuvuus Javan alustan (Java platform 2 Standard Editioon, J2SE) versioon 1.3 sekä Javan kehitystyökalun (Java Development Kit, JDK) versioon 1.3. Lisäksi OSGi-määritys versio 1.0 sisälsi jo joitain käyttäjien hallintaan liittyviä mekanismeja, joita ei haluttu poistaa versiosta 2.0.

Tulevaisuudessa kun Javan versioon 1.3 liittyvät puutteet, kuten käyttäjä perusteinen pääsyn valvonta pelkästään JDK 1.3 kanssa käytettäessä JAAS'ia, on poistettu voidaan JAAS toteuttaa OSGi-ympäristössä. Tämän jälkeenkin OSGi:n omaa käyttäjien hallintapalvelua tarvitaan ja se tulee tarjoamaan täydentäviä palveluita. Esimerkkinä tällaisesta on ryhmien jäsentietojen pysyvä säilyttäminen ja hallinta JAAS'in ulkopuolella, sillä JAAS ei hallitse pysyviä tietoja. Näin ollen käyttäjien hallintapalvelun rooli tulee muuttumaan enemmän taustatietokannaksi JAAS'ille.

3. PALVELUIDEN TUOTTAMINEN

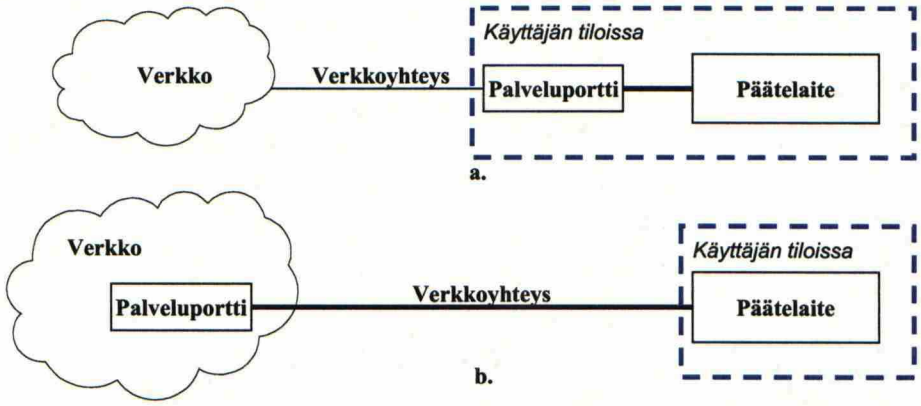
3.1 Operaattorin rooli

OSGi-ympäristön arkkitehtuuri voidaan helposti esittää arvoverkon avulla. Arvoverkossa esitetään erilaiset kaupalliset suhteet ympäristössä toimivien tahojen välillä. Seuraavassa kuvassa (Kuva 34) on piirretty OSGi-palvelun toimitukseen liittyvä arvoverkko verkossa olevalta palvelimelta asiakkaan kotona olevaan laitteeseen.



Kuva 34. Operaattorin arvoverkko OSGi-ympäristössä. [18]

OSGi määrittää kuinka palvelut saadaan välitettyä palvelun tarjoajalta asiakkaalle, muttei puutu toteutuksen teknisiin yksityiskohtiin. Operaattori voi toteuttaa toimitusketjun kahdella tavalla. Ensimmäinen tapa on toimittaa OSGi-palvelu asiakkaan kotona olevaan palveluporttilaitteeseen (Kuva 35a.). Tämä asettaa vaatimukseksi yhteyden ja riittävän tiedonsiirtokapasiteetin kodin ja operaattorin verkon välille. Toisena vaihtoehtona on sijoittaa OSGi-laitteisto operaattorin tiloihin ja toimittaa palvelu sieltä kotiin (Kuva 35b.). Tällöin asiakkaalle ei tarvitse toimittaa omaa palveluporttia vaan se on helposti ylläpidettävissä operaattorin tiloissa. Tällöin vaatimus kodin ja operaattorin verkon väliselle yhteydelle on suuremmat. Ruotsissa Telia on aloittanut OSGi-palvelupilottinsa käyttäen jälkimmäistä palvelun toimitusympäristöä.



Kuva 35. Kaksi palveluporttiverkon rakenne vaihtoehtoa.

Kuten kuvasta 34 havaitaan voidaan palvelun toimitus jakaa useiden toimijoiden osaprosesseihin. Todennäköisesti osa prosesseista toteutetaan yhdessä yrityksessä eli yritys toimii toimitusympäristössä useassa eri roolissa. Seuraavassa eri toimijoiden roolit on esitelty lyhyesti.

3.1.1 Operaattori

Operaattori on vastuullinen taho, joka vastaa koko toimintaketjun toiminnasta aina palvelun tarjoajasta asiakasrajapintaan saakka. Tänä päivänä olemassa olevista yrityksistä tämä tehtävä soveltuu ehkä luonnollisimmin erilaisille tele- ja verkko-operaattoreille.

3.1.2 Palvelun tarjoaja

Palvelun tarjoaja on kolmas osapuoli, joka on luonut OSGi palvelu*bundle*'in asiakkaan käytettäväksi. Tämä *bundle*, tai ryhmä *bundle*'eja, jotka toteuttavat sovelluksen, otetaan käyttöön OSGi palvelualustan hallintarajapinnan kautta joko asiakkaan tiloissa tai verkossa. Itse suoritusympäristön sijainti ei vaikuta palvelun tarjoajan tuottamaan palveluun, vaan toimii molemmissa. Palvelu on siis tässä yhteydessä kokoelma erilaisia ohjelmia, jotka toteuttavat yhdessä peruskäyttäjän laitteessa jonkin käyttäjän toivoman tehtävän eli palvelun. Esimerkiksi tällainen palvelu voi olla vaikka kodin hälytysjärjestelmän ohjaaminen.

Palvelun tarjoaja voi tuottaa jo olemassa olevaan palveluun lisäarvoa operaattorin kanssa lisäämällä tai mukauttamalla palvelun jakelukanavia. Esimerkkinä edellä mainitusta voisi olla tilanne, jossa palvelu käyttää operaattorin verkossa olevia

laitteita tai palveluita. Luonnollisesti, kun operaattorin ja asiakkaan välille on muodostunut asiakkuussuhde, on hyödyllistä palvelun tarjoajalle omata mahdollisimman läheiset välit operaattorin kanssa. Läheiset suhteet mahdollistavat sen, että palvelun tarjoaja mukauttaa palvelun vastaamaan paremmin operaattorin vaatimuksia. Vastaavasti operaattori voi tarjota kehitetylle palvelulle sopivia palvelun osia kuten esimerkiksi laskutuksen, taatun yhteyden, palveluiden kokoamisen ja vaihtoehtoisten tai rinnakkaisten jakelukanavien tukemisen. Tässä operaattori voi hyödyntää olemassa olevaa käyttäjähallintaansa.

Useiden pienten palvelun tarjoajien hallinta on usein hyvin kallista ja aikaa vievää. Tämän vuoksi operaattorin tulisikin olla varma yhteistyön kestävyydestä ennen yhteistyön aloittamista. Palvelun saatavuuden tulee myös olla jatkuva, jotta operaattorille ei tule ylimääräisiä ongelmia palvelun tarjoajan poistuessa markkinoilta.

3.1.3 Palvelun kokoaja

Palvelun kokoajan ja yhdistäjän roolit voidaan hyvin toteuttaa saman yrityksen sisällä, mutta näkyvät operaattorille erillisinä. Tämän takia ne on esitetty eriteltyinä myös tässä.

Palvelun kokoaja määrittelee asiakkaalle tehtävien palvelukokonaisuuksien toiminnot ja määrittelee ne *bundle*'it, joiden avulla nämä voidaan toteuttaa. Tämä voi vaatia *bundle*'ien hankkimista useilta palvelun tarjoajilta. Esimerkkipalveluna voidaan mainita kodin hallintapalvelu, joka sisältää seuraavat palvelukomponentit: turvallisuus, lämmityksen ohjaus ja kodin laitteiden kauko-ohjaus. Kun nämä komponentit hajotetaan vaatimuksiksi, voidaan joutua tilanteeseen, jossa etuoven valvontaa, uima-altaan pinnan tarkkailua ja muita tämän kaltaisia palveluita vaaditaan kodin hallintapalvelun toteuttamiseksi.

Operaattori voi itse suorittaa palveluiden kokoamisen, mutta kuten jo edellisessä kohdassa mainittiin on useiden pienten palvelun tarjoajien ja laitevalmistajien hallinta kallista ja aikaa vievää. Tämän vuoksi operaattorin kannattaa ostaa tämä palvelu ulkopuolelta. Palvelun kokoajan tulee pystyä osoittamaan kyvykkyytensä hallita useita pieniä yhtiöitä paketoidessaan tuotetta tai palvelua ja lisäksi ymmärtää

läheisesti operaattorin liiketoiminta. Kuitenkin tärkeimpänä vaatimuksena on osoitus pysyvyydestä markkinoilla. Tämä sen takia, että operaattorilla ei ole maineensa eikä liiketoimintansa puolesta varaa vaihtaa palvelun kokoajaa lyhyin väliajoin.

Palvelun kokoajalla tulee myös olla näkemys markkinoista niin, että se pystyy havaitsemaan, mitkä palveluntarjoajista tulevat säilymään markkinoilla. Lisäksi sillä tulee olla keino saada tarjottujen palveluiden oikeudet, jotta palvelun tarjoajan markkinoilta poistumisen jälkeen voidaan varmistaa palvelun katkoton käyttö. Palvelun kokoaja eräällä tavalla tarjoaa vakuutuksen palveluiden toimivuudesta riippumatta yksittäisistä palvelun tarjoajista.

3.1.4 Palvelun yhdistäjä

Palvelun yhdistäjä suorittaa teknisen osan palveluiden kokoamisesta. Kun kaikki palveluun tarvittavat komponentit on saatu kasaan, tarvitsevat ne vielä jonkin verran muutoksia toimiakseen yhdessä. Tarvittavan työn määrä riippuu paljon siitä, miten komponentit integroidaan ja testataan. Integrointi voi olla valittujen komponenttien hyväksymisestä aina uusien komponentteja integroivien *bundle*'ien tekemiseen ja testaamiseen. Yhtenä esimerkkinä voisi olla vaikka yhtenäistetty kommunikaatiopalvelu, jossa palveluun on yhdistetty eri toimittajien komponentteja kuten kalenteri, muistio, osoitteisto, tilanvaraus ja sähköposti. Kaikki komponentit on yhdistetty niin, että niiden käyttöliittymä on samanlainen. Näin helpotetaan palvelun käyttöä.

Jos vain mahdollista kokoajan ja yhdistäjän roolit tulisi yhdistää tai ainakin niillä tulisi olla molemmilla sama johto. Palvelun yhdistäjä voi olla kokoajan alihankija, samalla kun kokoaja ylläpitää jatkuvia suhteita palvelun tarjoajiin. Palvelun kokoaja voi käyttää useita palvelun yhdistäjiä palvelleessaan eri operaattoreita.

OSGi-markkinoiden tasaantuessa useat operaattorit todennäköisesti hoitavat sekä palvelun kokoajan että yhdistäjän roolit.

3.1.5 Verkkopalvelinoperaattori

Verkkopalvelinoperaattorin roolin soveltuu useimmille operaattoreille palvelukokonaisuuksien tarjoajan roolin lisäksi. Operaattorit ovat tottuneet

hallinnoimaan korkean käytettävyyden laitteita ja saaneet viime aikoina paljon kokemusta myös tietopalveluista Internet-palvelimien ylläpidon (Internet hosting) ja sähköisen kaupan aktiviteettien kautta. Tässä tapauksessa palvelimien ylläpitoon kuuluu alustojen (tekniikan ja ohjelmistojen) hallinnoimisen lisäksi tuen tarjoaminen sitä tarvitseville kolmansille osapuolille. Eräs tällainen voisi olla yritys, joka tarjoaa peruskäyttäjille paikan, josta hän voi ladata tarvitsemansa palvelun. Tästä toimenpiteestä yhtiö laskuttaa käyttäjää ja ylläpitää rekisteriä palvelusta ladatuista palveluista.

3.1.6 Palveluporttioperaattori

Palveluporttioperaattorin roolissa on yhtäläisyyksiä verkkopalvelinoperaattorin kanssa. Palveluportti on laite, jolle OSGi-palvelualusta on asennettu. Tässä yhteydessä palveluportin sijaintiin ei oteta kantaa. Se voi edelleen sijaita asiakkaan tiloissa tai operaattorin verkossa.

Palveluporttioperaattori tarjoaa asiakkaalle palveluporttilaitteita ja ylläpitää niitä riippuen asiakkaan valitsemasta omistus- ja asiakassuhteesta. Lisäksi palveluporttioperaattori voi tarjota asiakkailleen asennuspalveluita liittyen palveluporttiin ja siihen liitettäviin lisäominaisuuksiin esimerkiksi tietoturva.

3.1.7 Laitetoimittaja

Laitetoimittaja toimittaa laitteet, joita käytetään palveluiden toimittamisessa peruskäyttäjälle. Joissain tapauksissa olemassa olevia laitteita, kuten tietokoneita ja tv-sovittimia, voidaan käyttää palvelun esittämisessä ja niin palvelusidonnaisia laitteita ei tarvita. Asiakkaan tiloihin tarvitaan lisäksi laite muodostamaan tiedonvälityskanava asiakkaan ja palvelukeskuksen välille. Tällainen laite voi olla DSL- tai kaapelimodeemi.

Läheisesti laitteiden myyntiin ja jakeluun liittyy myös lähiverkon (LAN) laitteistojen tarjoaminen ja niiden hallinta. Monet operaattorit ovat ottaneet lähiverkkojen asentamisen osaksi heidän liiketoimintaansa. Näille operaattoreille ei kehittyneiden asiakkaan päätelaitteiden (CPE, Customer Premises Equipment) lisääminen tuotevalikoimaan ole vaikeaa.

Seuraava askel mentäessä lähemmäksi kodin laitteiden hallintaa on hyvin riippuvainen hallittavien laitteiden luonteesta. Monipuoliset, laajasti konfiguroitavat laiteet, kuten tietokoneet, ovat perustoimintoja lukuun ottamatta hankalia hallita, mutta OSGi-palvelualusta tarjoaa keinon kontrolloida ja hallita laitteita riippumatta niihin toimitetuista palveluista.

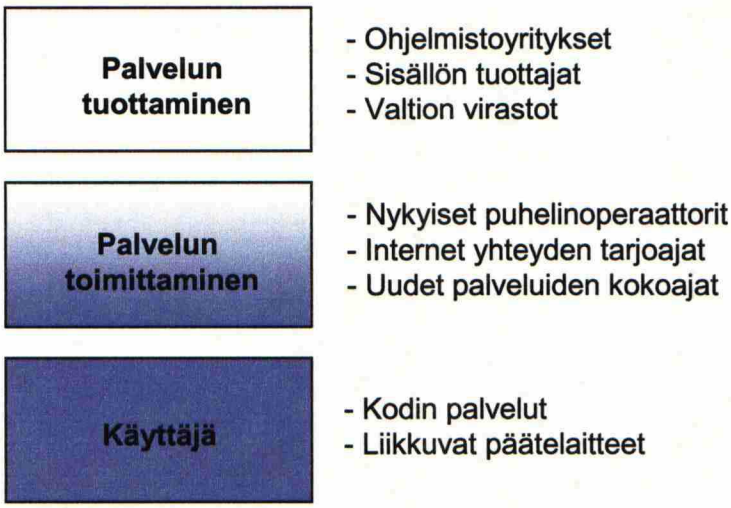
Laitteiden tulisi toimia OSGi-palvelualustan yhteydessä "liitä ja käytä" (Plug and Play) laitteiden tavoin. Tämä tarkoittaa sitä, että laitteen tulisi löytää itselleen sopivat ja tarvittavat ajurit OSGi-palvelualustasta laitteen käyttämiseksi. Ennen operaattorin tuli liittoutua laitevalmistajan kanssa varmistaakseen yhteensopivuuden palvelun ja muun verkon kanssa. OSGi-palvelualusta määrittää yleiset laiteominaisuudet ja näin laitteen vaihtaminen onnistuu ilman, että palvelu*bundle*'ia tarvitsee päivittää. Pahimmassa tapauksessa OSGi-palvealustassa oleva ajuri on kuitenkin päivitettävä. Tämä kuitenkin tapahtuu automaattisesti ja ajuri voidaan hakea esimerkiksi laitevalmistajan Internet-sivustoilta.

Todellisuudessa on epätodennäköistä, että edellä kuvattu tilanne toteutuu lähitulevaisuudessa, vaan useimmat palvelut tulevat käyttämään sovelluskohtaisia CPE-laitteita palvelun käyttöä varten.

Jokainen kuvassa 34 esitetty toimija voidaan yhdistää useiden toisten toimijoiden kanssa, jotta palveluiden toimittaminen on kannattavaa operaattorille. Ennen uuden palvelun tullessa markkinoille kolmannet osapuolet huolehtivat suurimmasta osasta toimintoja ja (puhelin)operaattorit hoitivat alustan ja avustivat yhdistämistehtävissä. Aikojen saatossa operaattorit omaksuivat osan toiminnoista tai kaikki toiminnot oman organisaationsa hoidettavaksi ja optimoivat näin prosesseja ja vähensivät kustannukset minimiin. Tilanne on todennäköisesti samankaltainen OSGi:n käyttöönoton yhteydessä, koska OSGi:n laitetekniikka on jo ennestään tuttua operaattorin kannalta. Jotta näin voisi käydä, tulee OSGi-ympäristöön muodostua seuraavat toimijat: palvelun tarjoajat, palvelun kokoajat, palvelun yhdistäjät sekä laitetoimittajat. Tällä hetkellä laitevalmistajat ovat jo olemassa ja palvelun tarjoajia alkaa pikku hiljaa ilmestyä markkinoille.

3.2 Palveluiden toimittaminen asiakkaalle

Palvelun toimitusketju voidaan jakaa karkeasti kolmeen osaan: palveluiden tuottaminen, palveluiden toimittaminen ja palvelun asiakkaat (Kuva 36).



Kuva 36. Palvelun toimitusketju jaettuna kolmeen vaiheeseen.

Palvelun tuottajaryhmässä olevat yritykset tai yhteisöt voivat keskittyä vain palvelun kehitykseen ilman että heidän tarvitsee miettiä palvelun toimittamista asiakkaalle. Palvelun tuottaja on sopinut palveluiden jakelusta palvelun kokoajien kanssa, jotka kuuluvat seuraavaan tasoon eli palvelun toimittajiin. Palvelun tuottaja voi olla ohjelmointiyrityksen lisäksi myös sisällön tuottaja. Hyvä esimerkki on uutistoimisto tai vaikka sääennustuksia tekevä yritys. Näin esimerkiksi uutistoimisto saa uuden kanavan, mitä kautta se voi jakaa jo kertaalleen tuottamaansa sisältöä.

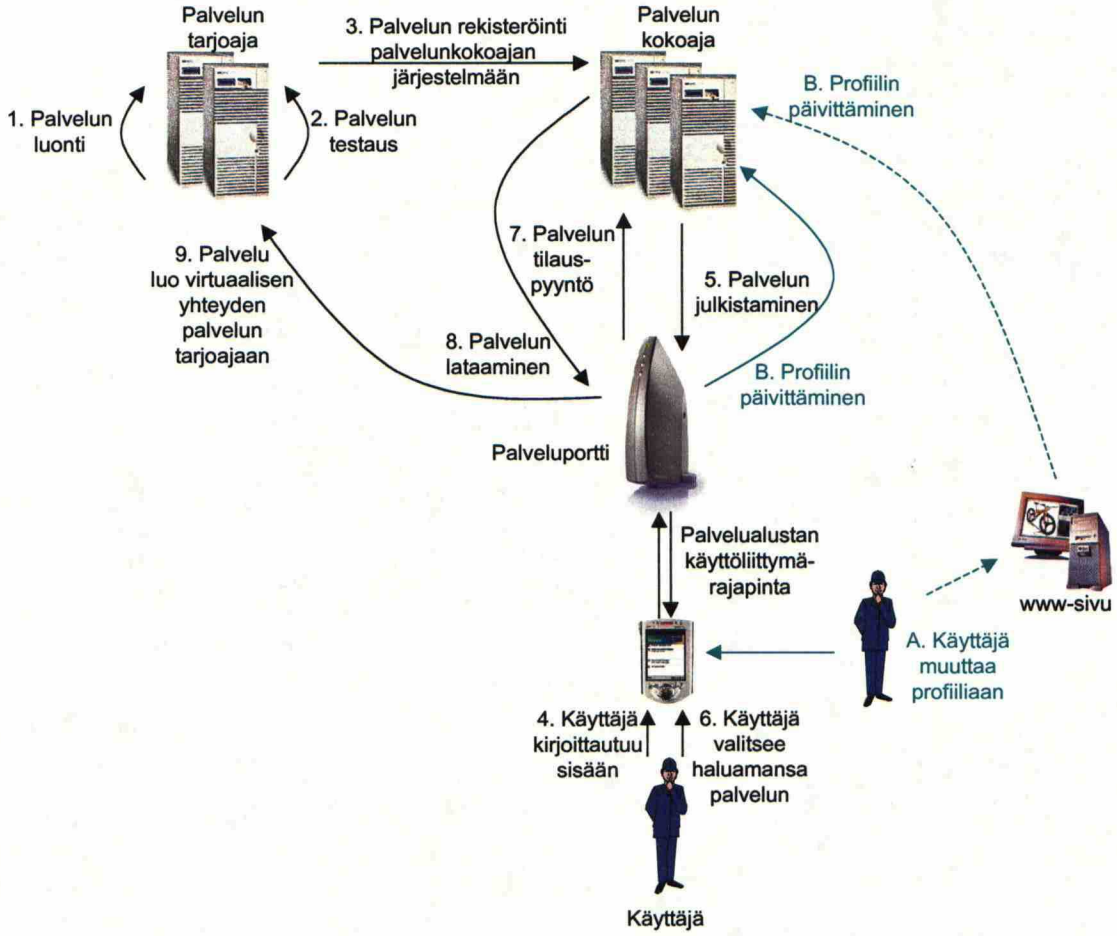
Asiakkaalle eli käyttäjälle tämä näkyy edullisempina käyttökustannuksina, kun palvelusta maksetaan esimerkiksi vain käytön ajalta. Lisäksi asiakkaalle voidaan tarjota laajempaa valikoimaa palveluista kuin aiemmin on ollut taloudellisesti mahdollista. Näin käyttäjä voi valita omia tarpeita parhaiten vastaavan palvelun.

Palvelun toimittajaportaalle uusi OSGi-palveluporttiympäristö tuo haasteita, kuinka hallita hajautettua palvelualustaverkkoa. Palveluiden lataaminen tarvittaessa vaatii käyttäjää hankkimaan sopivan yhteyden palveluiden kokoajan palvelimelle. Usein tällainen yhteys on hyvä olla jatkuva, jolloin kyseeseen tulevat esimerkiksi DSL-tekniikat (Digital Subscriber Line) ja kaapelimodeemi. Jos palvelun toimittaja on

puhelinoperaattori, tarjoaa OSGi-palvelualusta operaattorin asiakkaille uusia mahdollisuuksia käyttää hyväkseen operaattorilta ostettua yhteyttä.

Jotta palvelualustojen jakelu olisi kannattavaa ja tuotteiden hinnat saataisiin pysymään kohtuullisina, tulee laitteen asennus olla mahdollisimman helppoa. Tämä tarkoittaa sitä, että normaalitilanteessa asiakas voi itse asentaa laitteen eikä palveluporttioperaattorin asentajan tarvitse käydä ollenkaan asiakkaan tiloissa. Todennäköisesti asiakas tarvitsee kuitenkin jonkin verran apua yhteyden kytkemisessä ennen kuin palveluportti voi ottaa yhteyttä verkkoon.

Jos yhteys on jo olemassa, voidaan palveluportti varustaa eräänlaisella ensikäynnistysohjelmalla (Initial Provisioning). Tämä ohjelma sisältää tiedon siitä, mistä verkon osoitteesta löytyy palveluportin asentamiseen tarvittavat konfiguraatiotiedot. Asiakkaalle voidaan tarjota useita eri vaihtoehtoja esimerkiksi 'käytä olemassa olevaa yhteyttä (ADSL) ja ota yhteys Elisan Palveluporttipalveluun' tai 'käytä olemassa olevaa yhteyttä (ADSL) ja ota yhteys WipeSec-palveluporttipalveluun'. Tämän jälkeen palveluportti ottaa yhteyden valittuun palveluporttipalveluun ja lataa operaattorin ennalta valitsemat yhteiset palvelut sekä toimittaa asiakkaan tiedot operaattorin tietokantaan. Samalla kun yhteisiä palveluita asennetaan palveluporttiin, asiakas saa listan muista palveluporttipalvelun sisältämistä palveluista hintoineen. Näistä palveluista asiakas voi valita haluamansa ja ne asennetaan palveluporttiin. Seuraavassa esimerkissä käsitellään tarkemmin palvelun toimittamista palvelun tarjoajalta asiakkaalle.



Kuva 37. Palvelun toimitusketju palvelun tarjoajalta asiakkaalle.

Kuvassa 37 on esitetty palvelun toimitusketju kokonaisuudessaan. Palvelun tarjoaja tuottaa uuden palvelun (1.) ja testaa omassa testausympäristössään sen yhteensopivuuden muiden OSGi-ympäristön palveluiden kanssa (2.). Kun palvelun tarjoaja on varmistunut palvelunsa toiminnasta, rekisteröi se sen palvelun kokoajan palvelurekisteriin (3.).

Tässä esimerkissä asiakas käyttää palveluporttia erillisen päätelaitteen avulla. Tällainen päätelaite voi olla tavallinen kämmentietokone (Personal Digital Assistant, PDA). Käyttäjä kirjautuu palveluporttiin (4.) ja hakee palvelun kokoajalta listan tarjolla olevista palveluista (5.). Tämä voidaan myös automatisoida niin, että aina kun palvelun kokoajalla on uusi asiakkaan määrittelemään profiiliin sopiva palvelu, toimitetaan asiakkaalle siitä eräänlainen mainos.

Palvelulistalta käyttäjä valitsee tarvitsemansa palvelun (6.) esimerkiksi reittipalvelun ja pyyntö palvelusta välitetään palvelun kokoajalle (7.). Palvelu ladataan asiakkaan palveluporttiin (8.) ja käynnistetään. Nyt palvelu on asiakkaan käytettävissä.

Käyttäjä on lähdössä matkalle ja kaipaa neuvoa mikä olisi paras reitti päästä määränpäähän. Hän syöttää tarvittavat tiedot reittipalveluun. Palvelu muodostaa yhteyden palvelun tarjoajan tietokantaan (9.) ja hakee parhaimman reitin asiakkaan valitsemaan määränpäähän sekä lataa reittiin liittyvät kartat.

Palveluportissa on asiakkaan tiedot sisältävä profiili. Tätä profiilia käyttäjä voi muokata päätelaitteella palveluportissa tai sitten hän voi käyttää hyväkseen erillistä palvelun kokoajan www-sivua, johon hän voi syöttää tietoja muutoksista (A.). Profiili sisältää tietoja palveluportin käyttäjästä, hänen mieltymyksistään sekä palveluporttiin asennetuista laitteista. Palveluportti toimittaa muutokset asiakkaan profiilissa palvelun kokoajalle (B.). Profiilin tietoja käyttäen palvelun kokoaja osaa suodattaa palvelulistalta asiakkaan konfiguraatiossa toimimattomat sekä asiakasta kiinnostamattomat palvelut pois.

Kuva 38. Palvelun poistaminen palveluportista.

Kun käyttäjä palaa matkalta, hän voi poistaa palvelun palveluportista (Kuva 38 kohta 10.). Palveluportti toimittaa palvelun peruutuspyynnön palvelun kokoajalle (11.), joka poistaa palvelun palveluportista (12.). Näin palveluportissa on vain asiakkaan haluamat ja tarvitsemat palvelut. Kun käyttäjä jossain vaiheessa haluaa taas käyttää reittipalvelua, tilaa hän sen uudestaan palvelun kokoajan ylläpitämältä palvelulistalta (Kuva 37. kohta 6.).

3.3 Palvelulaskutus

OSGi-palvelualustaympäristössä laskentatietoa muodostuu kaikilla toimitusketjun eri tasoilla. Laskutuksen ajatellaan yleisesti jakautuvan kolmeen eri ryhmään: kiinteä kuukausittainen laskutus, kertalaskutus ja sisältöön perustuva laskutus.

IP-verkoissa peritään tällä hetkellä yleisesti kiinteitä ja yhteyden kaistanleveyteen perustuvia hintoja. Käyttöaikaan perustuva laskutusmalli sopii huonosti IP-verkkoon. IP-protokolla on suunniteltu yhteydettömän liikenteen välittämiseen, ja näin yhteyden keston mittaaminen IP-verkossa on ongelmallista.

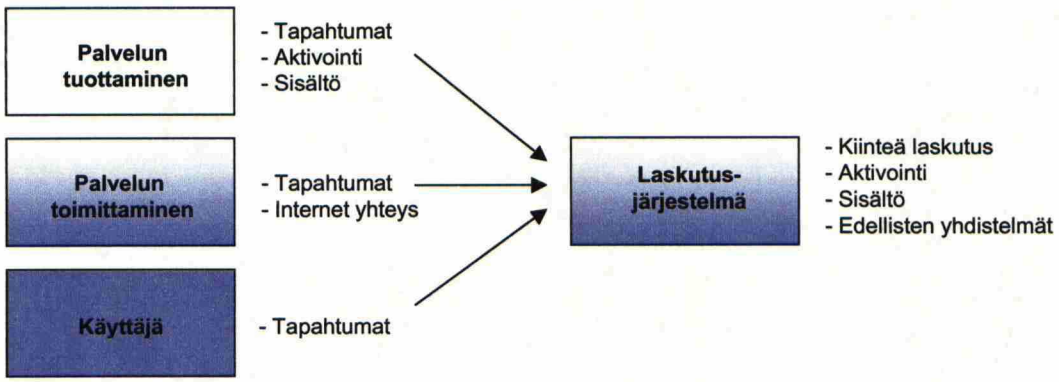
Kiinteässä kuukausilaskutuksessa asiakas saa kerran kuukaudessa, tai jonkin muun sovitun ajan välein, laskun. Lasku kattaa kaikki sopimuksen piirissä olevien palveluiden käytön, joten asiakas tietää tarkalleen laskunsa suuruuden. Tällöin laskenta ei mittaa resurssien käyttöä ja näin suosii paljon verkkoa kuormittavia palveluita. Tämä vastaavasti johtaa yhteyden tarjoajan tarpeeseen tehdä investointeja lisätäkseen verkon kapasiteettia. Toisaalta kiinteä veloitus ei edellytä tarkkaa laskentatietojen keräämistä verkosta, mikä osaltaan säästää operaattorin kustannuksia.

Kertalaskutuksessa lasku syntyy, kun asiakas esimerkiksi tilaa uuden palvelun palveluporttiin. Esimerkkinä tästä voisi olla tilausmusiikkipalvelu, jossa kappale maksetaan kerran ja sen jälkeen se tallennetaan palveluporttiin, josta sen voi kuunnella aina haluttaessa.

Sisältölaskutus on nimensä mukaisesti riippuvaista palvelun sisällöstä, esimerkiksi uutistoimisto voi hinnoitella uutisilleen hinnan sen mukaan kuinka tuoreita ne ovat. Tällöin mitataan enemmän asiakkaan kokemaa hyötyä kuin itse tiedon siirtoa. Vastaavasti edellä esimerkkinä ollut reittipalvelu voi laskuttaa tilatuista kartoista ja reitistä asiakasta erikseen. Selvästi voidaan nähdä, että tulevaisuudessa sisällöstä velottaminen muodostuu itse datansiirrosta laskuttamista tärkeämmäksi.

Näitä eri laskutusmuotoja voidaan vapaasti yhdistää, jolloin voidaan laskuttaa palvelun tilaamisesta, aktivoinnista ja sisällöstä erikseen. Hyvä esimerkkikohde voisi olla vaikka videopuheluiden uudelleen ohjaus. Palvelun tarjoaja laskuttaa, kun asiakas ostaa itselleen palvelun. Kun asiakas aktivoi palvelun, laskutetaan häneltä

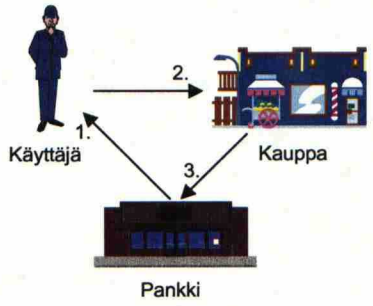
jokin pieni summa ja kaikista uudelleen ohjatuista yhteyksistä asiakas maksaa palveluluokan ja siirrettävän tietomäärän tai keston mukaan. Pitää kuitenkin huomioida, että myös liiketoimintaympäristö, kilpailijat jne., vaikuttavat laskutusmallin valintaan, ei pelkästään tekniset tekijät. Kuvassa 38 on esitetty erilaisten laskentatietojen keräys toimitusketjun kerroksista.



Kuva 39. Laskutuksen muodostuminen toimitusketjun eri tasoilla.

Itse laskutus voi olla ongelmallista, jos kyseessä on pieni palveluiden tarjoaja. Kun palvelun hinta on muutamia euroja, ei ole taloudellisesti kannattavaa lähettää palvelun ostaneille asiakkaille erillistä laskua. Ratkaisun tähän ongelmaan tuovat sähköinen raha (eCash, mikromaksut micropayments), luottokorttimaksut, erilaiset pankkien suoramaksupalvelut, kuten Nordean Solo-maksu, sekä operaattorin tarjoamat laskutuspalvelut.

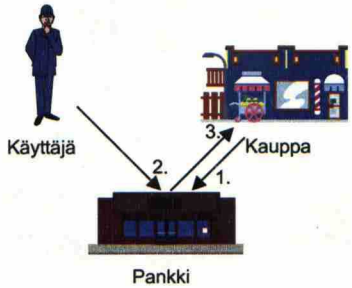
Sähköisestä rahasta puhutaan, kun asiakas ottaa yhteyttä pankkiin tai muuhun sähköisen rahan jakelijaan saadakseen kolikoita. Tässä vaiheessa pankki nostaa asiakkaan tililtä kolikoita vastaavan summan. Nämä kolikot tallennetaan asiakkaan tietokoneelle erilliseen sähköiseen kukkaroon. Tämän jälkeen asiakas ottaa yhteyden kauppiaaseen ja maksaa ostoksensa pankista saamillaan kolikoilla. Kauppiaalle kolikot hyvitetään hänen palauttaessaan ne pankkiin. Alla oleva kuva (Kuva 40) kuvaa rahan kulkua pankista asiakkaan kautta kauppiaalle ja takaisin pankkiin. Sähköisen rahan ongelmana on se, miten pankki huomaa, jos samaa kolikkoa käytetään moneen kertaan.



Kuva 40. Rahan kulkema reitti sähköisen rahan tapauksessa. [20]

Luottokorttimaksu eroaa sähköisestä rahasta ja suoramaksupalvelusta siten, että siinä rahaliikenteen riskin ottaja on luottokortin myöntäjä esimerkiksi pankki tai luottolaitos. Muuten maksu on yhtä suoraviivainen kuin tavallisessa laskukaupassa. Ongelmia luottokorttimaksussa tuottaa lähinnä tilaajan varmentaminen. Tällä hetkellä on suhteellisen helppo vihamielisen tahon saada tietoonsa luottokortin numero ja käyttää sitä hyväkseen Internetissä tehdyissä kaupoissa.

Suoramaksupalvelussa pankki toimii välittäjänä asiakkaan ja kauppiaan välillä. Sen jälkeen kun asiakas on valinnut haluamansa tuotteen, maksaa hän sen pankin tarjoamalla suoramaksupalvelulla. Tässä palvelussa kauppias toimittaa pankille ostoksen yhteissumman ja oman tilinumeronsa. Näistä tiedoista pankin suoramaksupalvelu tuottaa asiakkaalle laskun, jonka asiakas voi maksaa käyttäen hyväkseen pankkinsa tarjoamaa verkkopankkia ja sen tunnuksia. Näin kauppias saa maksun ostetusta tavarasta heti ja voi toimittaa tuotteen asiakkaalle ilman, että kauppiaan tarvitsee seurata tuotteen maksua (reskontra). Seuraava kuva (Kuva 41) selventää suoramaksupalvelun toimintaa.



Kuva 41. Suoramaksupalvelun toiminta asiakaan ostaessa tuotteita kauppiaalta. [20]

Ongelmaksi suoralaskupalvelun osalta muodostuu se, että kauppiaan täytyy tehdä sopimus kaikkien rahalaitosten kanssa tarjotakseen tasapuolisen palvelun kaikille asiakkailleen. Tämä syö nopeasti palveluiden myynnistä saatavaa voittomarginaalia, koska pankit veloittavat tapahtumakohtaisen maksun lisäksi kiinteää palvelumaksua suoramaksupalvelun käytöstä.

Laskutuspalvelu sopii hyvin operaattorin tarjoamaan palvelupakettiin. Operaattoreilla on jo olemassa valmis laskutusjärjestelmä ja –suhde. Heille pienienkin maksujen laskuttaminen ei ole kannattamatonta toimintaa, sillä asiakkaalle lähetetään joka tapauksessa lasku esimerkiksi yhteydestä Internetiin. Tällöin operaattori voi laskuttaa asiakasta palvelun tarjoajan puolesta niin, että kaikki asiakaan verkossa suorittamat transaktiot liitetään osiksi yhteyslaskua. Yleisesti operaattori voi olla, kuten jo mainittiin, olemassa oleva puhelinoperaattori tai erillinen laskutusoperaattori. Tämän vuoksi käytetään jatkossa edellä mainituista operaattoreista yleisesti nimitystä laskentaoperaattori.

Tavanomaisen laskutuksen kanssa on myös mahdollista tarjota asiakkaalle erilaisia laskun rajoituspalveluita, joilla asiakas voi seurata laskun muodostumista. Toinen vaihtoehto on niin sanotun ennalta maksettu palvelumaksu (Pre-Paid), missä asiakas maksaa operaattorille etukäteen tietyn summan. Käyttäessään palveluita verkossa tätä ennalta maksettua vähennetään kunnes se loppuu ja asiakkaalle lähetetään asiasta viesti. Tämän jälkeen asiakas ei voi käyttää verkon maksullisia palveluita ennen kuin hän on tallentanut lisää rahaa palvelutililleen.

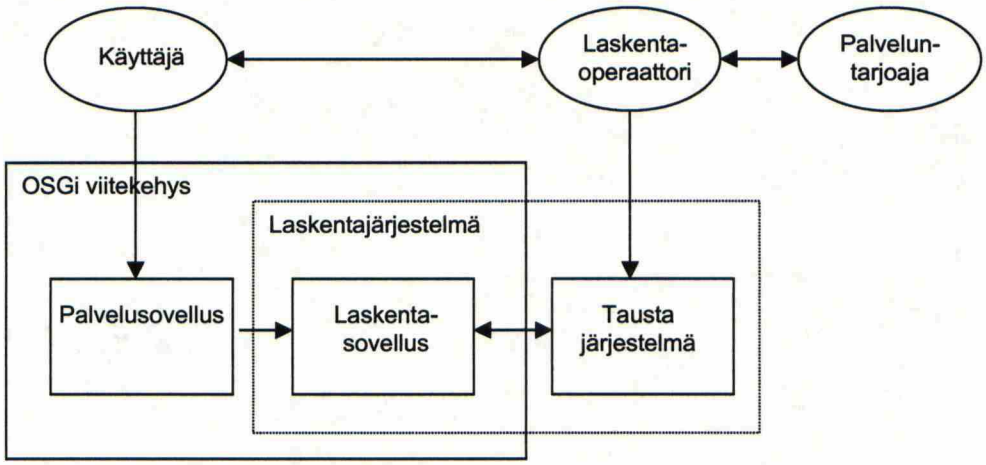
Samoin kuin suoramaksupalvelu vaatii tämäkin malli kauppiaan ja operaattorin välisen sopimuksen. Mutta jos palvelualustaympäristöstä halutaan tehdä varmatoiminen, on operaattori joka tapauksessa tehnyt sopimuksen palveluntarjoajien eli kauppiaiden kanssa palveluiden tarjoamisesta palvelualustoille operaattorin verkossa. Tähän samaan sopimukseen voidaan hyvin liittää kauppiaan sopimus laskutuksen hoitamisesta operaattorin laskentajärjestelmien kautta.

3.4 Laskentapalvelu

Pohjimmiltaan laskentajärjestelmät perustuvat laskentatapahtumien keräämiseen. Mekanismi voidaan suunnitella erilailla riippuen eri tahojen välisestä

luottamusmallista sekä tietoturvan ja tapahtumien käsittelyn tehokkuuden välisestä optimoinnista. Jotta tapahtumia voitaisiin käsitellä edullisesti, perustuu useat tietoliikennejärjestelmien laskentamekanismit laskentatietueiden (charge detail record, CDR) keräämiseen. Tällä hetkellä OSGi-palvelualustaan ollaan määrittelemässä yhteistä laskentapalvelua, joka keräisi laskentatietoa OSGi-viitekehyksestä operaattorin laskentajärjestelmälle. Tämä tapahtuu niin, että laskentapalvelu luo laskentatapahtumia (charging event, CE), jotka sitten kerätään ja käsitellään laskentaoperaattorin puolesta.

OSGi-viitekehys voidaan toteuttaa erilaisille alustoille, eikä *bundle* tavallisesti voi tietää mille alustalle viitekehys on toteutettu. Tämä johtaa siihen, että palvelusovellus pitää suoritusympäristöä oletusarvoisesti epäluotettavana. Kuvassa 42 on esitelty havainnollisesti, miten laskentapalvelu sijoittuu ja keiden kanssa sillä on luottamussuhde. Alusta pitäen suunnittelussa on lähdetty siitä oletuksesta, että laskentapalvelu liittyy osana operaattorin olemassa olevaan laskentajärjestelmään. [21]



Kuva 42. Laskentapalvelun sijainti viitekehyksessä ja siihen liittyvät luottamussuhteet. [21]

Kuten jo luvun alussa mainittiin, on luottamus muihin instansseihin laskentapalvelun tärkein vaatimus. Laskentamekanismi perustuu käytännössä kolmeen luottamussuhteeseen. Ensimmäinen oletus on että palvelun tarjoaja luottaa tiettyihin laskentaoperaattoreihin. Palvelusovellus luottaa vain luotetulta laskentaoperaattorilta vastaanottamiinsa tietoihin.

Toiseksi palvelusovelluksen tarjoaja olettaa, että laskentaoperaattori tarkistaa palvelun käyttäjän valtuutustiedot ennen kuin käyttäjän sallitaan käyttää palvelua. Jos käyttäjällä ei ole oikeita oikeuksia käyttää palvelua, ilmoittaa laskentaoperaattori tästä palvelulle. Käytännössä tämä tarkoittaa sitä, että palvelun käyttäjällä on joko suora tai epäsuora yhteys laskentaoperaattoriin.

Viimeinen oletus on, että laskentaoperaattori luottaa sellaisen sovelluksen lähettämiin tietoihin, joita hallitsee tietty palvelun tarjoaja.

Yllä esitetty luottamusmalli ei sisällä palvelualustan operaattoria, mutta suunniteltu mekanismi käyttää sertifikaatteja ja/tai salaisia avaimia, jotka luovutetaan vain luotettujen palvelualustaoperaattoreiden palvelualustojen käyttöön.

Teknisesti laskentapalvelu perustuu kaksivaiheiseen neuvottelumekanismiin. Palvelun laskentatapahtuma lähetetään laskentapalvelulle ja palvelu jää odottamaan laskentapalvelun vastausta siihen ennen kuin palvelu jatkaa toimintaansa. Laskentapalvelun vastaus voi kestää monestakin syystä. Esimerkiksi ostettaessa palvelu verkosta käyttäen siihen tarkoitettua sovellusta. Ennen kuin sovellus voi toimittaa palvelun, se tarkistaa, että käyttäjällä on sopivat valtuutustiedot sovelluksen käyttöön. Koska palvelu monesti toimitetaan ilman takeita yhteyden laadusta (best effort), voi yhteyden tai palvelimen palvelun laatu heikentää palvelua siinä määrin, ettei siitä ole käyttäjälle mitään hyötyä, esimerkiksi palvelun lataaminen ei onnistu. Tällöin ei käyttäjää laskuteta epäonnistuneesta palvelun hankinnasta.

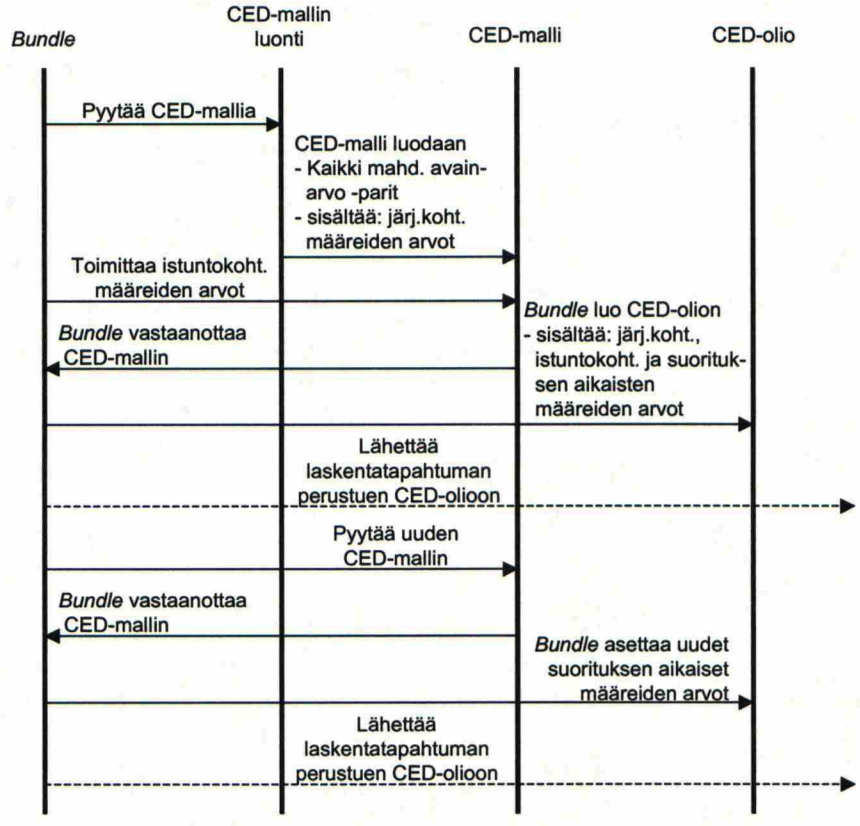
Laskentatapahtuma eroaa Javan tapahtumasta, koska jokainen laskentatapahtuma nähdään tapahtumana sekä toimituksen luotettavuuden huomioimisessa että tietoturvassa koskien kohdetta ja lähdettä.

Laskentatapaus kuvataan laskentatapahtuma kuvauksena (charge event descriptor, CED). CED on esimääritelty avain-arvo –parien joukko, jossa on yksi tai useampi määre. Määreet ja niiden lukumäärä vaihtelevat sovelluksesta riippuen, ollen tiedonsiirtosovelluksissa tyypillisesti lukumääräisesti varsin suuri. Laskentapalvelun toiminta perustuu CED:n sisältämien arvojen kuvaaman järjestelmän tilan analysointiin. Suurin osa CED:n sisältämistä arvoista kuitenkin säilyy

muuttumattomina peräkkäisissä laskentatapahtumissa, eikä *bundle* voi muuttaa kaikkia CED:n sisältämiä arvoa. Näillä toimenpiteillä pyritään varmistamaan laskennan eheys.

CED:n sisältämät määreet voidaan jakaa kolmeen ryhmään: järjestelmäkohtaisiin määreisiin (system), istuntokohtaisiin määreisiin (session) ja suorituksen aikaisiin määreisiin (run-time). Ryhmät kuvaavat hetkeä jolloin ryhmään kuuluvat määreet asetetaan. Järjestelmäkohtaiset määreet asetetaan ennen kuin *bundle* vastaanottaa CED-mallin eikä *bundle* voi muuttaa määreiden arvoja. Järjestelmäkohtaiset määreet ovat yhteisiä kaikille laskentatapahtumille.

Istuntokohtaiset määreet asettaa *bundle*, mutta ne ovat voimassa koko istunnon ajan. Nämä määreet asetetaan CED-malliin. CED-malli määrittelee ne avain-arvo-parit, jotka CED voi sisältää. Ensimmäisen CED-mallin mukaan luodun instanssin jälkeen, *bundle* ei voi muuttaa määreiden arvoja ilman uuden CED-mallin luomista. Vastaavasti suorituksen aikaiset määreet voivat muuttua milloin tahansa (Kuva 43).

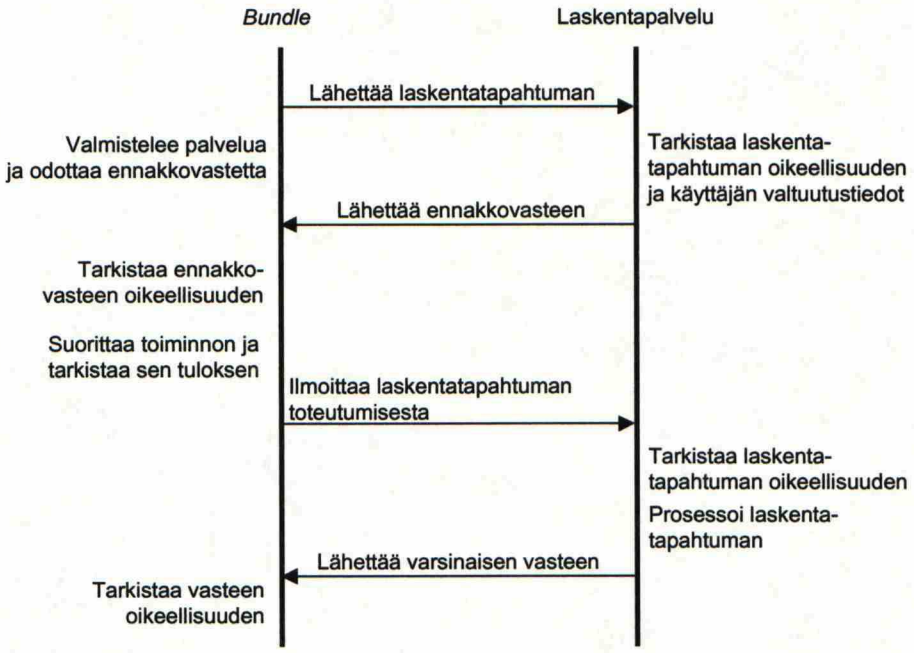


Kuva 43. Laskentatapahtuman valmistelu [21]

Määreet ovat jaettu eri ryhmiin, jotta määreille voidaan tarjota yksinkertainen suojamekanismi. Järjestelmäkohtaiset määreet usein kuvaavat järjestelmän perustietoja ja pääsy näihin tietoihin on usein rajoitettu komponenteille, joilla on järjestelmän hallitsijan oikeudet. Istuntokohtaiset määreet voivat määritellä jonkin loogisen yhteyden peräkkäisten laskentatapahtumien välille. Suojaamalla istuntokohtaiset määreet voidaan laskentatapahtumien oikeuttamaton muuttaminen estää.

Bundle toimii yhteistyössä laskentapalvelun kanssa laskentapalvelun rajapinnan kautta. Laskentatapahtuma voidaan lähettää vain maksuistunnon (PaymentSession) yhteydessä. Maksuistunto on laskentapalvelun turvamekanismin ydin, se muodostuu palvelun tarjoajan ja laskentaoperaattorin kätellessä.

Laskentatapahtuma koostuu kaksivaiheisesta tiedonsiirrosta, jossa laskentaoperaattori vastaanottaa laskentatapahtuman ennen kuin *bundle* toteuttaa palvelupyynnön (Kuva 44). Kun laskentaoperaattori vastaanottaa laskentatapahtuman, tarkistaa se siitä autentikoinnin ja käyttäjän valtuutustiedot, ennen kuin palauttaa ennakkovasteen *bundle*'ille. Autentikointi laskentapalvelun ja *bundle*'in välillä perustuu välitettävien tietojen salaukseen sovitulla salaustekniikalla esimerkiksi jaetun salaisuuden tai julkisen ja salaisen avaimen menetelmään. *Bundle* vastaanottaa laskentaoperaattorin palauttaman ennakkovasteen ja tarkastaa hyväksyikö laskentaoperaattori toiminnon ja päättää suorittaako palvelupyynnön vai ei. Palvelupyynnön toteutuksen jälkeen *bundle* lähettää laskentaoperaattorille ilmoituksen laskentatapahtuman toteutumisesta ja operaattori palauttaa *bundle*'ille varsinaisen vasteen, minkä mukaan tilaajaa on laskutettu.

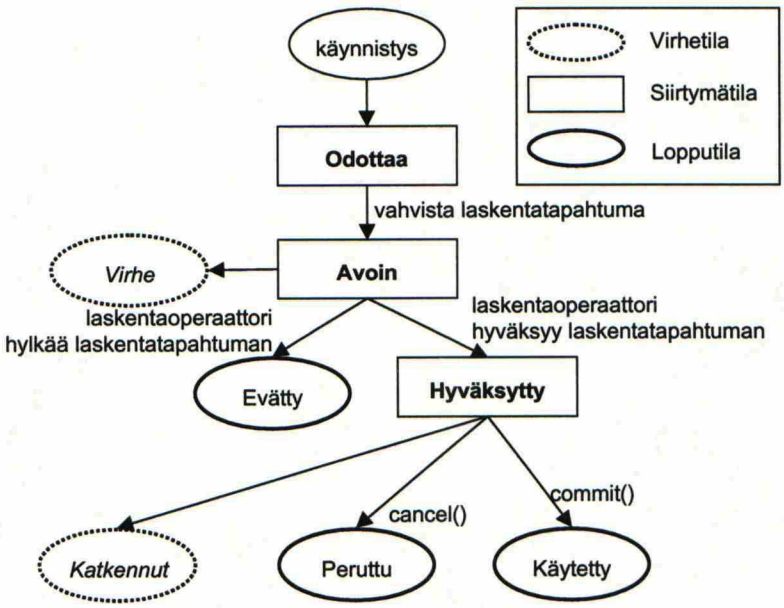


Kuva 44. Laskentatapahtuman kaksi eri vaihetta, oikeus palvelun käyttöön ja itse laskutus. [21]

Järjestelmä*bundle* voi tarkkailla ja hallita tietyllä tasolla laskentatapahtumien virtaa siitä lähtien, kun laskentatapahtuma lähetetään ja kunnes laskentapalvelu palauttaa varsinaisen vasteen. Kuva 45 kuvaa laskennan tilamallin. Mallissa aloitustilaa vastaa odottaa (waiting) -tila, missä laskentaoperaattori ei ole vielä vahvistanut laskentatapahtumaa. Heti kun laskentaoperaattori vastaanottaa laskentatapahtuman, muuttuu laskentatoiminnon tila avoimeksi (open).

Tässä vaiheessa laskentaoperaattori tekee päätöksen laskentatapahtuman hylkäämisestä tai hyväksymisestä. Jos laskentatapahtuma hylätään, siirtyy laskentatoiminto tilaan evätty (denied), muuten siirrytään tilaan hyväksytty (accepted).

Laskentatoiminnon tilaan vaikuttavista päätöksistä seuraavan tekee *bundle*. Jos se haluaa jatkaa palvelupyynnön suorittamista, käyttää (commit) *bundle* laskentapalvelulta saamansa ennakkovasteen ja laskentatoiminto siirtyy tilaan käytetty (committed). Jos *bundle* kuitenkin päättää olla käyttämättä ennakkovasteen ja peruu palvelupyynnön, siirtyy laskentatoiminto tilaan peruttu (cancelled).



Kuva 45. Laskentaan liittyvien toimintojen tilamalli. [21]

Tilamallissa on kaksi virhetilaa, jotka ilmaisevat sen, että laskentapalvelu ei osaa sanoa, minkä valinnan laskentaoperaattori on tehnyt koskien laskentatapahtumaa. Katkennut (broken) –tila kuvaa tilannetta, jossa laskentapalvelu ei löydä avointa laskentayhteyttä (Kuva 44) ja ei sen vuoksi voi suorittaa laskentatehtävää loppuun. Tila voi syntyä esimerkiksi silloin, kun laskentapalvelu joutuu odottamaan liian pitkään *bundle*'in päätöstä laskennan hyväksymisestä tai hylkäämisestä. Tällöin laskentaoperaattori tulkitsee laskentatoiminnon keskeytyneeksi ja siirtyy tilaan katkennut. Toinen virhetila kuvaa tilannetta, missä laskentapalvelussa tapahtunut virhe estää palvelun luotettavan läpiviemisen.

Laskentapalvelu kohtaa useita uhkia roolinsa vuoksi palveluporttiympäristössä. Seuraavassa on lueteltu madollisista uhista vakavimmat. Yhden uhan muodostavat vihamielisen käyttäjän luomat tai muokkaamat *bundle*'it, jotka voivat toteuttaa pyydetyn palvelun ilman laskentapalvelun mukana oloa. Tällöin kaikki muutettujen *bundle*'ien käynnistämät palvelut jäävät kirjautumatta laskentatietokantaan.

Toinen uhka on, että käyttäjä muuttaa *bundle*'in määrittelemiä määreiden arvoja, ja näin muuttaa esimerkiksi palvelun maksajaksi jonkun muun kuin itsensä. Tähän lopputulokseen voidaan myös päästä toisella tavalla. Tällöin käyttäjä huijaa *bundle*'in allekirjoittamaan vihamielisen *bundle*'in luoman viestin ja näin saa jonkun toisen tahon tavallaan hyväksymään itsensä palvelupyynön maksajaksi.

Kolmantena uhkana voidaan mainita salausmenetelmien yleinen uhka, jossa salausavain tai -avaimet jostain syystä joutuvat vihamielisen tahon haltuun. Tällöin avaintenhallintamekanismissa tulee olla menetelmä, millä voidaan sulkea tietyt avaimet pois käytöstä. Kahden ensimmäisen uhan ratkaisuksi OSGi:ssä on mietitty tarpeeksi vahvan salauksen käyttöönottoa sekä laskentapalvelun ja *bundle*'in välisessä viestinnässä että laskennan kanssa tekemisissä olevien *bundle*'ien ohjelmakoodissa.

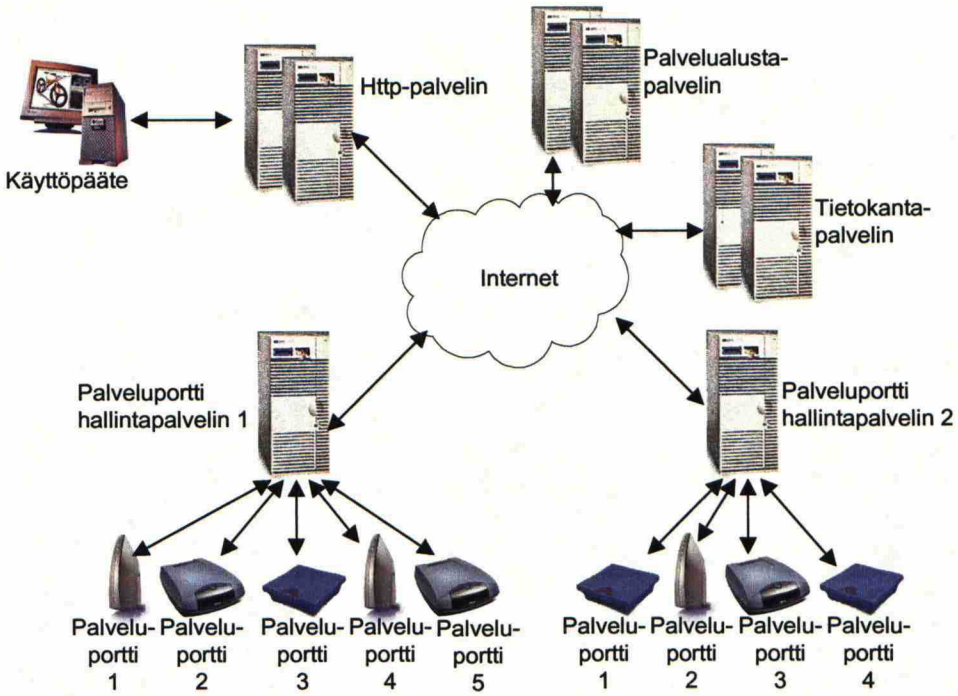
3.5 Hajautetun palveluporttiverkon hallinta

Palveluporttikonsepti pitää sisällään ajatuksen käyttäjän kannalta huolettomasta käytöstä, sallien hänen tehdä päätöksiä palveluiden tilaustasolla. Tällöin käyttöjärjestelmän versioista huolehtiminen, ajurin versioiden valinta, ohjelmistopakettien riippuvuussuhteiden hallinta jne. jää jonkun muun vastuulle. Toisaalta palveluportilta odotetaan myös korkeaa toimintavarmuutta. Kaupallisen operaattorin tarjotessa asiakkailleen palveluporttilaitteita edellä esitellyistä vaatimuksista syntyy operaattorille tarve etähallita ja tarkkailla palveluportteja sekä niihin asennettuja palveluita ja laitteita.

3.5.1 Verkon rakenne

Palveluporteilla voidaan muodostaa kaksi toisista poikkeavaa verkkorakennetta. Toisessa verkkomallissa palveluportti sijaitsee asiakkaan tiloissa, kun toisessa palveluportti sijaitsee palveluporttioperaattorin tiloissa. Tällöin asiakkaan tiloihin tulee asentaa erillinen verkkopääte, joka sovittaa asiakkaan laitteen operaattorin tiloissa olevaan palveluporttiin.

Molemmissa tapauksissa verkon hallinta muistuttaa läheisesti toisiaan. Seuraavassa kuvassa (Kuva 46) on esitelty yksi mahdollinen verkkomalli, missä jokainen asiakas on vastaanottanut oman palveluporttilaitteen ja se on asennettu asiakkaan tiloihin.



Kuva 46. Hajautetun palveluporttiverkon rakenne.

Palveluportit ovat jaettu loogisiin ryhmiin esimerkiksi palveluportin valmistajan, sijainnin tai ominaisuuksien mukaan. Palveluportti hallintapalvelin on yhteydessä palveluportteihin. Hallintapalvelin määritellään jokaiselle palveluportille silloin, kun palveluportti liitetään verkkoon.

Tietokantapalvelimeen tallennetaan jokaisen palveluportin käyttäjien tiedot sekä tiedot ostetuista ja asennetuista palveluista. Näitä tietoja voivat hakea hallintapalvelimet sekä verkon hallitsija. Verkkoa hallitaan http-palvelimeen tehdyn www-sivun kautta. Sivulla voidaan esittää vapaasti erilaisiin hallintatilanteisiin sopivia tietoja. Kun tietoja päivitetään, tallennetaan ne samalla sekä tietokantapalvelimelle että niihin palveluportteihin, joita tietojen päivitys koskee. Tarvittaessa palveluporttien hallitsija voi myös saada erilaisia listauksia asiakkaille asennetuista palveluista sekä määrätä tiettyjen palveluiden asennus ennalta määriteltyihin palveluportteihin. Laskentatiedot voidaan kerätä http-palvelimelle, mutta jotta laskentatoiminnallisuus olisi mahdollisimman suojattua, tulisi laskentaa varten olla oma palvelin.

Palvelut ovat saatavissa palvelualustapalvelimelta, johon palvelun yhdistäjä toimittaa testatut palvelut. Palvelualustapalvelin myös tuottaa siihen taltioiduista

palveluista listan, jonka avulla palveluportin käyttäjä voi tilata itselleen uusia palveluita tai päivittää vanhoja.

Kuvassa esitelty käyttöpääte toimii operaattorin rajapintana verkon hallintaan. Palveluporttien hallintakonsolissa eli käyttöpäätteellä voi myös suorittaa suurempia eräajoja käyttäen konekielisiä syöttötiedostoja. Näin operaattori voi helposti muuttaa valitsemiensa palveluporttien asetuksia helposti ja nopeasti ilman, että sen tarvitsee tehdä samat muutokset yksitellen kaikkiin valittuihin palveluportteihin. Samalla operaattori voi ajastaa päivityksen tapahtuvaksi öiseen aikaan, jolloin päivitys ei vaikuta palveluportin käytettävyyteen havaittavasti.

3.5.2 Palveluportin asentaminen ja ylläpito

Palveluportin asennus on hieman ongelmallinen. Kuinka tarjota asiakkaille kustannustehokkaasti mahdollisuus käyttää palveluporttia? Kuten jo aiemmin tässä luvussa on mainittu, voi operaattori tarjota asiakkailleen keskitettyä palveluporttia, missä palveluportti fyysisesti sijaitsee operaattorin tiloissa esimerkiksi puhelinkeskuksen yhteydessä. Tällöin asiakkaalle on kuitenkin toimitettava erillinen verkkopääte ja sovitin, jonka avulla asiakas voi liittää haluamansa laitteet palveluporttiin ja näin käyttää hyväkseen palveluportin tarjoamia edistyneitä palveluita.

Jos kyseessä on yksinkertainen sovitinlaite, osaa asiakas todennäköisesti itse asentaa laitteen. Tämä tarkoittaa lähinnä tilanteita, jossa asiakkaan tarvitsee kytkeä virtapistoke seinään ja verkkokaapeli esimerkiksi ADSL-modeemiin. Monimutkaisemmissa tilanteissa, kuten jos asiakas on ostanut myös murtohälytyspalvelun, on todennäköistä, että operaattorin tulee teettää laitteen asennus asiantuntevan henkilön toimesta.

Ongelmat ovat samat myös siinä tapauksessa, että palveluportti on asiakkaan tiloissa oleva laite. Tällöin myös voidaan tarjota asiakkaalle mahdollisuus itse asentaa laite, mutta monimutkaisissa tilanteissa on parempi käyttää suoraan asiantuntevan asentajan apua.

Toinen valintatilanne tulee eteen, kun asiakkaan laite vikaantuu. Tuleeko operaattorin lähettää silloin oma asentajansa paikalle vai toimittaako asiakas laitteen

operaattorin valtuuttamaan huoltoon? Nämä ovat ratkaisuja, joita operaattorin tulee miettiä ennen kuin aloittaa kaupallisen toiminnan käyttäen palveluportteja. Yksiselitteistä vastausta on vaikea antaa, mutta todennäköisesti yksinkertaiset laitteet asiakas voi toimittaa suoraan huoltoon ja monimutkaisempien laitteiden yhteyteen operaattori voi myydä asiakkaan tarpeen mukaan räätälöityjä huoltosopimuksia.

Huoltosopimuksesta ja vian laadusta riippuen päätetään, lähteekö operaattorin huoltomies käymään asiakkaan tiloissa vaihtamassa palveluportin uuteen vai viekö asiakas laitteen huoltoon. Huoltosopimuksia voidaan tarjota myös yksinkertaisempien laitteiden haltijoille.

3.5.3 Palveluportin hallinta

Palveluportti muodostuu palveluporttilaitteesta ja siihen asennetuista palveluista. Operaattorille on tärkeää, että se pystyy kustannustehokkaasti hallitsemaan jokaista sen verkkoon kytkettyä palveluporttia. Toisaalta kotiin asennettu palveluportti vaatii myös perhekohtaista hallintaa esimerkiksi käyttäjien luomista, niiden oikeuksien määrittelyä, palveluiden tilaus jne. Tällaiset tehtävät voidaan tehdä operaattorin toimesta, mutta tämä voidaan kokea asiakkaan taholta liian hitaalta ja vaivalloiselta tavalta hallita palveluporttia. Tämän vuoksi palveluporttiin voidaan tarjota asiakkaalle hallintarajapinta, minkä avulla hän voi lisätä kodin palveluporttiin käyttäjiä ja määritellä niiden paikalliset käyttäjäoikeudet operaattorin sallimissa rajoissa. Näin palveluportin hallintatehtävät voidaan jakaa kahteen ryhmään: ulkoiseen ja sisäiseen hallintaa.

Ulkoisella hallinnalla tarkoitetaan niitä hallintatoimenpiteitä, jotka operaattori suorittaa palveluportille. Tällaisia toimintoja ovat esimerkiksi laitteen verkkoasetusten määrittäminen, kaikkiin palveluportteihin asennettavien palveluiden hallinta ja erilaisista vikatilanteista selviäminen. Jotta asiakas kokee ulkoisen hallinnan turvallisena, tulee asiakkaan ja operaattorin välillä olla luottamussuhde. Teknisesti tämä vaatii varmaa käyttäjän tunnistusta hallintarajapinnalla sekä vahvasti salattua yhteyttä.

Sisäinen hallinta koostuu niistä hallintatoimenpiteistä, joita käyttäjällä on oikeus tehdä hallussaan olevassa palveluportissa. Käyttäjä voi valita haluaako hän hallita itse palveluporttia, vai ottaako hän yhteyttä operaattoriin silloin, kun palveluportin asetuksia tarvitsee muuttaa. Näin palveluportin käyttö ja hallinta voidaan tehdä mahdollisimman helpoksi asiakkaalle.

Toistaiseksi OSGi ei ole saanut määritystä OSGi-palvelualustan etähallinnasta valmiiksi, vaan se on lähinnä kerännyt hallintatarpeita lähtökohdaksi määritelmän tekoa varten. Määritystyötä tekevällä etähallintaryhmällä on tavoitteena saada määritys valmiiksi vuoden 2002 loppuun menneessä. [22]

4. PALVELUESIMERKKI OSGI:N KÄYTÖSTÄ KOTIVERKOISSA

4.1 Tavoite

Palvelualusta antaa eri osapuolille mahdollisuuden tarjota uudenlaisia palveluita asiakkaalle. Tällä hetkellä asiakkaalla voi olla useita laitteita, jotka tekevät vain sille laitteelle kuuluvia tehtäviä. Laitteiden yhteen liittäminen vaatii kalliita ratkaisuja tai on muuten vain mahdotonta.

Palveluesimerkin tarkoituksena on kokeilla kuinka kotiympäristössä voitaisiin käyttää hyväkseen edullisia kodin automaatiolaitteita ja tässä työssä käsiteltyä OSGi-palvelualustaa. Lähtökohtana on, että palvelu toteuttaa OSGi:n avoimuusperiaatteen eli palvelu voidaan suorittaa missä tahansa OSGi-palvelualustassa. Samalla palvelu toteutetaan muutenkin OSGi:n ohjeistuksen mukaisesti.

Tavoitteena esimerkillä on saada konkreettinen palvelu, joka käyttää toista laitetta apunaan toteuttaakseen palvelun ja näin saada kokemusta palveluportin toiminnasta vähintään yhden palvelun käytössä.

Palveluesimerkiksi päätettiin ottaa kotiin sopiva edullinen hälytyspalvelu, jossa liiketunnistimen havaitessa liikettä valvottavalla alueella palveluportti taltioi kuvan kameralla ja ilmoittaa liikkeestä kodin omistajalle

4.2 Lähtökohdat

Palvelun toteutus käynnistyi sopivan OSGi-palvelualustan valinnalla. Sellainen löytyi Siemens Metavectorilta. Heidän PyliX-laitteensa on ensisijaisesti tarkoitettu koti- ja pientoimistokäyttöön sisältäen puhelinliitännät, USB-väylän, sarja- ja rinnakkaisporttit sekä Ethernet-liitännät ulkoiselle ja sisäiselle verkolle. Teknisesti ottaen PyliX on palvelinympäristö, johon OSGi-palvelualusta on toteutettu tukemaan uusien palveluiden tuomista valitulle kohderyhmälle. Palvelualustan lisäksi PyliX sisältää tulostus-, tiedosto-, sähköposti- ja DHCP-palvelimen ollen näin varsin houkutteleva ratkaisu pienen kotona toimivan yrityksen tarpeisiin.

Siemensin laitteeseen oli toteutettu muutama esimerkkipalvelu, joita yhdessä Siemensin suunnittelijoiden kanssa muutettiin vastaamaan olemassa olevia laitteita ja palveluvaatimuksia. Palveluun liittyvien sähkölaitteiden ohjaukseen valittiin edullinen ja helposti asennettava X-10-teknologia. Tehon ja lämpötilan mittaus päätettiin toteuttaa käyttäen hyväksi hieman arvokkaampaa Echelonin kehittämää tieto- ja automaatiojärjestelmää LON'ia (Local Operating Network). Syy miksei mittauspalvelua toteutettu X-10-tekniikalla on X-10 protokollan yksinkertaisuus. X-10-protokolla ei tue tiedonsiirtoa muuten kuin ohjauskäytössä. Tämän vuoksi valinta kohdistui LON-laitteisiin. Koska muutokset esimerkkipalveluihin kohdistuivat lähinnä X-10 tekniikkaan liittyviin palveluihin ei LON-teknologiaa käsitellä tarkemmin.

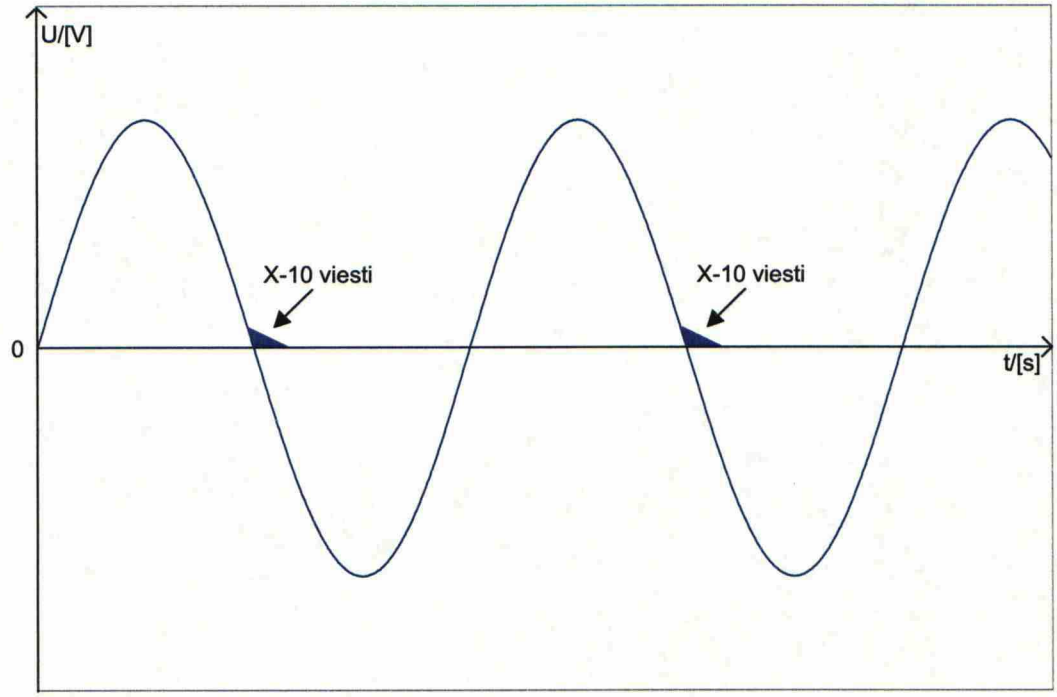
4.3 X.10

X-10 protokollan on alkujaan kehittänyt 1975 Pico Electronics. Protokolla on suunniteltu X-10-lähettimen ja vastaanottimen väliseen kommunikointiin olemassa olevan sähköverkon kautta. Lähettimet ja vastaanottimet kytketään pistorasioihin, sähkökeskukseen tai seinäkojeiden tilalle.

X-10 on kommunikointikieli sähkölaitteiden kauko-ohjaukseen. Tämä yksinkertainen ohjausjärjestelmä mahdollistaa mm. kodin valojen ja laitteiden ohjauksen kodin olemassa olevaa sähköverkkoa hyväksi käyttäen. Kustannuksia lisäävää uudelleenjohdotusta ei täten tarvita, kuten monissa muissa kotiautomaatiojärjestelmissä esimerkiksi Echelonin LON-teknologiassa. Tämä lisää asennustyön nopeutta ja helppoutta. X-10 protokollassa on käytettävissä 256 eri osoitetta. Jos useamman laitteen halutaan reagoivan samaan viestiin, valitaan kojeisiin sama yksikkönumero. Kaikkia X-10 yhteensopivia laitteita ja kojeita voidaan vapaasti yhdistellä keskenään.

Lähettimet ja vastaanottimet kytketään pistorasioihin, mutta sähkökeskukseen asennettavat laitteet vaativat ammattitaitoisen sähköasentajan. Lähettimet ohjaavat vastaanottimia komennoilla päälle, pois tai himmennä, riippuen ohjattavasta laitteesta. Jokaisella X-10 laitteella on oma osoitteensa. Vastaanottimet eivät reagoi käskyihin, joita ei ole osoitettu niille. Oikea koodi oikeassa X-10 laitteessa aiheuttaa halutun toiminnon.

X-10 signaali lähetään sillä hetkellä kun vaihtosähköverkon hetkellisjännite on nolla (Kuva 47). Suurin osa X-10 viesteistä ovat jännitteen huippuarvoltaan alle yhden voltin suuruisia. X-10 vastaanottimet kykenevät toimimaan, jos signaalin taso on vähintään 100 mV. Näin ollen X-10 signaalit eivät aiheuta häiriöitä suurimmalle osalle kodin sähkölaitteista. Jos vaihtosähköverkossa on liikaa häiriöitä, tämä saattaa aiheuttaa ongelmia viestin kulkemisessa. Jos häiriöt ovat verkossa suurempia kuin X-10 signaali, silloin viesti ei kulje perille vastaanottimeen. Tähän ongelmaan on kehitetty avuksi suotimet. Myös herkimmät laitteet kannattaa suojata erillisillä suodattimilla, mitkä poistavat X-10 ohjaussignaalin verkkovirrasta. Vastaavasti samoja suodattimia kannattaa käyttää sellaisten laitteiden yhteydessä, jotka aiheuttavat häiriöitä verkkoon. Tällaisia laitteita ovat esimerkiksi tietokoneiden hakkuritehonlähteet ja oikosulkumoottorit. [23]



Kuva 47. X-10 signaalin sijainti vaihtosähköverkossa. [23]

X-10-viestejä voidaan lähettää vain yksi kerrallaan. Jos kaksi signaalia lähetään samalla hetkellä, ne törmäävät ja viesti ei kulje perille. Tällä törmäyksellä on vaikutusta vain jos signaaleilla on eri koodit esimerkiksi, kun lähetetään koodit: Talokoodi A yksikkönumero 2, sekä talokoodi A ja yksikkönumero 4. Jos

signaaleilla on samat koodit, vastaanotin ei ota huomioon toista signaalia ja viesti menee perille.

Lähettimet lähettävät 3 eri peruskomentoa, jotka ovat päälle, pois ja himmennä, perustuen X-10 osoitteeseen (talokoodi ja yksikkönumero). Jokaiseen X-10 moduliin valitaan osoite valintakytkimillä, joista ensimmäinen on talokoodi (HOUSE), ja toinen on Yksikkönumero (UNIT). Talokoodi voidaan valita väliltä A-O ja yksikkönumero väliltä 1-16. Tavallisesti kodin eri kerroksiin valitaan eri talokoodit, eli alakertaan valitaan A ja yläkertaan B. Tämä kuitenkin aiheuttaa sen, että jokainen kerros on varustettava omalla vastaanottimella, jotta kaikkia kerroksia voidaan hallita esimerkiksi kaukosäätimellä.

Laitteita joilla ohjataan järjestelmää kutsutaan lähettimiksi. X-10 laitteita voidaan ohjata muutamilla eri tavoilla. On olemassa laitteita jotka muuntavat infrapunaviestin X-10 signaaleiksi. Radioaalloilla toimivilla kaukosäätimillä voidaan ohjata X-10 laitteita useiden kymmenien metrien etäisyydeltä. Pöytäohjaimet ovat suosittu tapa laitteiden ohjaukseen. Niissä on yksinkertaiset painikkeet haluttujen X-10 kojeiden ohjaukseen. Lisäksi on olemassa X-10 ajastimia, joilla voidaan ohjelmoida yksinkertaisia tiettyyn aikaan suoritettavia tapahtumia. Tässä on vain osa laitteista, joilla on mahdollista ohjata X-10 laitteita. Ohjaimien määrä kasvaa koko ajan.

X-10 laitetta, joka reagoi lähettimen antamaan käskyyn, kutsutaan vastaanottimeksi. Yksinkertaisin vastaanotin on pieni pistotulpalla varustettu moduli, joka asennetaan pistorasiaan. Kojemodulissa on sisäänrakennettu rele joka, ohjaa virran päälle tai pois, lähettimen antamasta X-10 viestistä riippuen. Valaisinmoduli on vastaavanlainen kuin kojemoduli, mutta releen tilalla käytetään virran säätelyyn sopivaa triac'ia. Tämä mahdollistaa päälle ja pois komentojen lisäksi myös himmennystoiminnon.

Muita vastaanottimia ovat normaalien kytkimien tilalle asennettavat X-10 kytkimet ja himmentimet, itse valaisimeen tai laitteeseen asennettavat pienet X-10 valaisin- tai kojemodulit. Myös kiskokiinnitteisiä koje- ja valaisinmoduleita on mahdollista asentaa talon sähkökeskukseen. Tämä seikka kannattaa ottaa huomioon jo sähkösuunnitteluvaiheessa.

Kaksisuuntaisella X-10 laitteella tarkoitetaan X-10 kojetta, joka osaa kertoa oman tilansa. Tietokonesovitin on yksi tällaisista laitteista. Tietokonesovittimella voidaan ohjata X-10 laitteita kotona tai etäkäyttönä vaikka työpaikalta. Sovitin mahdollistaa myös erilaisten makrojen rakentamisen ja käytön. Tietokonesovitin on X-10 järjestelmän aivot, jolla saadaan kaikki hyöty irti kotiautomaatiosta. Se kytketään pistorasiaan, jonka kautta se kykenee kommunikoimaan lähettimien ja vastaanottimien kanssa sekä ottamaan komentoja vastaan radio-ohjausyksiköltä.

X-10 laitteiden toiminta aluetta rajoittaa lähinnä ohjaussignaalin vaimeneminen 100 mV:n alle sekä kodin sähköistyksen toteuttaminen käyttäen tasaisesti kaikkia kolmea vaihetta. Toiminta-aluetta voidaan laajentaa toistimilla ja vahvistimilla. Toistin on kiskokiinnitteinen X-10 moduli, joka yhdistää vaiheet niin että X-10 signaali kykenee kulkemaan kaikissa kolmessa vaiheessa. Vahvistin on vastaavasti laite, joka nostaa X-10 signaalin voimakkuutta jopa 20-kertaiseksi normaaliin tasoon nähden. Useimmissa suuremmissa järjestelmissä tätä laitetta olisi suotavaa käyttää, jotta signaali kulkisi myös kauimmaisille X-10 laitteille.

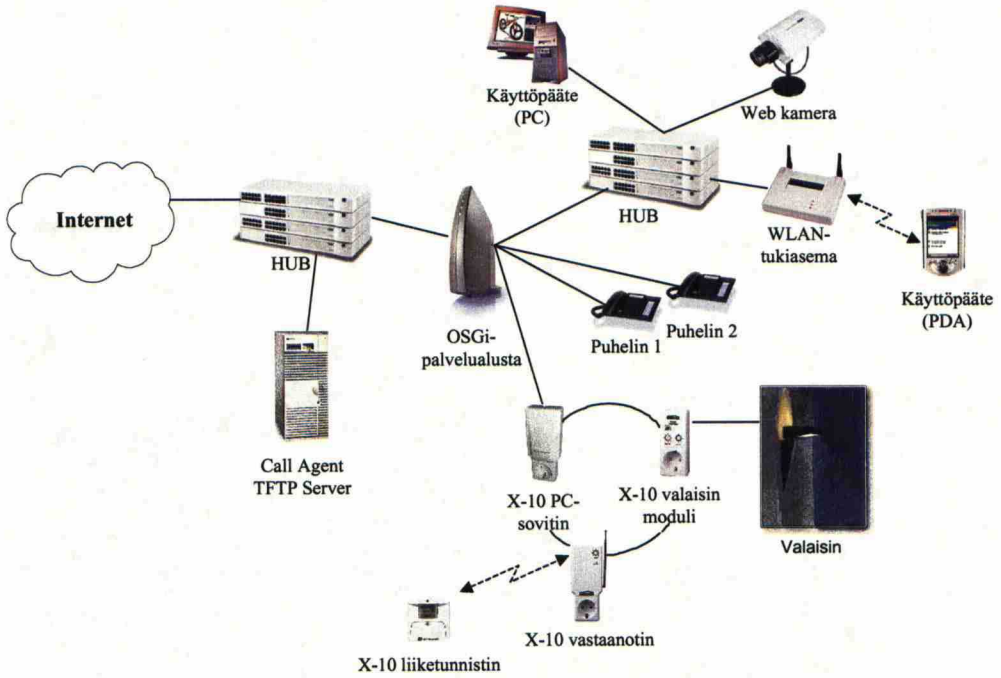
4.4 Palvelun toteutus

Esimerkkipalvelu toteutettiin kuvassa 48 esitetyillä komponenteilla. Palveluesimerkin keskeinen komponentti on itse OSGi-palvelualusta, johon on liitetty X-10-tietokonesovitin, puhelimia ja keskitin. Keskittimen avulla palvelualustaan liitettiin verkkokamera, WLAN-tukiasema ja paikallinen käyttöpääte. WLAN-tukiaseman luoman radioverkkosolun avulla kämmenmikroon on voitu luoda liikkuva käyttöpääte palvelualustalle. Kaikki palvelualustan tarjoamat ohjelmalliset palvelut ovat myös käytettävissä kämmenmikrosta.

X-10-tietokonesovittimella hallitaan kaikkia X-10-verkkoon liitettyjä laitteita. Esimerkkipalvelun tapauksessa X-10-verkkoon liitettiin kaksi valaisinmoduulia, joihin liitettiin tavalliset hehkulamppuvalaisimet. Tämän lisäksi X-10-verkossa on vastaanotin, joka vastaanottaa langattoman liiketunnistimen lähettämät signaalit.

Edellä mainitut komponentit muodostavat palveluportin sisäverkon. Nämä laitteet ovat käytettävissä ja hallittavissa tietyin myöhemmin esiteltävin poikkeuksin vain

kodin ja WLAN-radioverkosolun kantaman sisäpuolella. Toinen keskitin, palvelin ja Internet muodostavat kodin ulkopuolisen verkon.



Kuva 48. Esimerkkipalveluun liittyvien laitteiden muodostama verkko.

Kuvassa esiintyvä Call Agent eli puhelun ohjauspalvelin ja yksinkertainen tiedoston siirtopalvelin (Trivial File Transfer Protocol, TFTP) hoitavat esimerkkipalvelussa käytetyn OSGi-palvelualustan tarvitsemia tukitoimintoja. Käynnistyessään palvelualusta vaatii erillisen lataustiedoston, missä ilmenee seuraavat tiedot: käytettävä ohjelmistoversio, hallintarajapinnalle saapuvien kutsujen sallitut IP-osoitteet, palveluportin hallintatunnus ja salasana sekä eri palvelimien IP-osoitteet. Tämän tiedoston palvelualusta lataa tiedostopalvelimelta. Alla on lyhyt ote palvelualustan käynnistyslokista, josta ilmenee käynnistystiedoston lataaminen ja sen käsittely. Tiedostossa on tiedot kahdesta käyttäjästä `'ElisaUser1'` ja `'ElisaUser2'`, jotka palvelualusta luo hallintarajapinnalle.

```
<0 00:00:05.180><BootHandler::ValidAcfUrl::True><Sending the acf-url to the acf
parser><OBJ=8069C070>
Notice    02008 [May 22, 2002 08:22:30:000] - Start parsing the ACF-file
                                              elisa.acf from server
                                              192.168.2.19
Notice    02501 [May 22, 2002 08:22:30:000] - ACF - Parsing started
Notice    03001 [May 22, 2002 08:22:31:000] - RGW_AuthLevel: Adding user
                                              'ElisaUser1' to the webserver
Notice    03002 [May 22, 2002 08:22:31:000] - RGW_AuthLevel: Adding user
                                              'ElisaUser2' to the webserver
Notice    02502 [May 22, 2002 08:22:32:000] - ACF - Parsing finished
```

Valittu palvelualusta tukee myös mahdollisuutta tehdä IP-puheluita tavallisella analogisella puhelimella, mitä varten esimerkkipalveluverkkoon lisättiin puhelun ohjauspalvelin hoitamaan puhelun välityksessä tarvittavaa MGCP-protokollaa (Media Gateway Control Protocol).

Kun palvelun ohjelmakoodia tarkastellaan voidaan havaita, että esimerkkipalvelu koostuu muutamasta oleellisesta *bundle*'ista: alarm, axisnetcam, mail ja x10. *Bundle*'it on nimetty selkeästi niiden tehtävän mukaisesti. Alarm-*bundle* vastaa palvelun toteutuksesta ja käyttää toisten *bundle*'ien tuottamia palveluita. Vastaavasti axisnetcam-*bundle* tuottaa valvontakuvaa, mail-*bundle*'in avulla voidaan lähettää ilmoitus hälytyksestä sähköpostitse käyttäjän haluamaan sähköpostiosoitteeseen ja x-10 –*bundle* tarjoaa laitteiden ohjauspalvelun.

Liitteessä C olevasta ohjelmakoodista nähdään hälytyspalvelun yksinkertaistettu toiminta. Palvelualustaan on määritelty erillinen asetustiedosto, josta hälytyspalvelu saa tarvitsemansa tiedot. Nämä tiedot asetetaan palvelualustassa tavallisella tekstieditorilla. Tämä tarkoittaa käytännössä sitä, että operaattorin tulee tehdä asetukset palveluporttiin joko asennuksen yhteydessä tai sitten erikseen ottamalla yhteys palveluporttiin. Alla on esimerkki palveluportin asetustiedostosta.

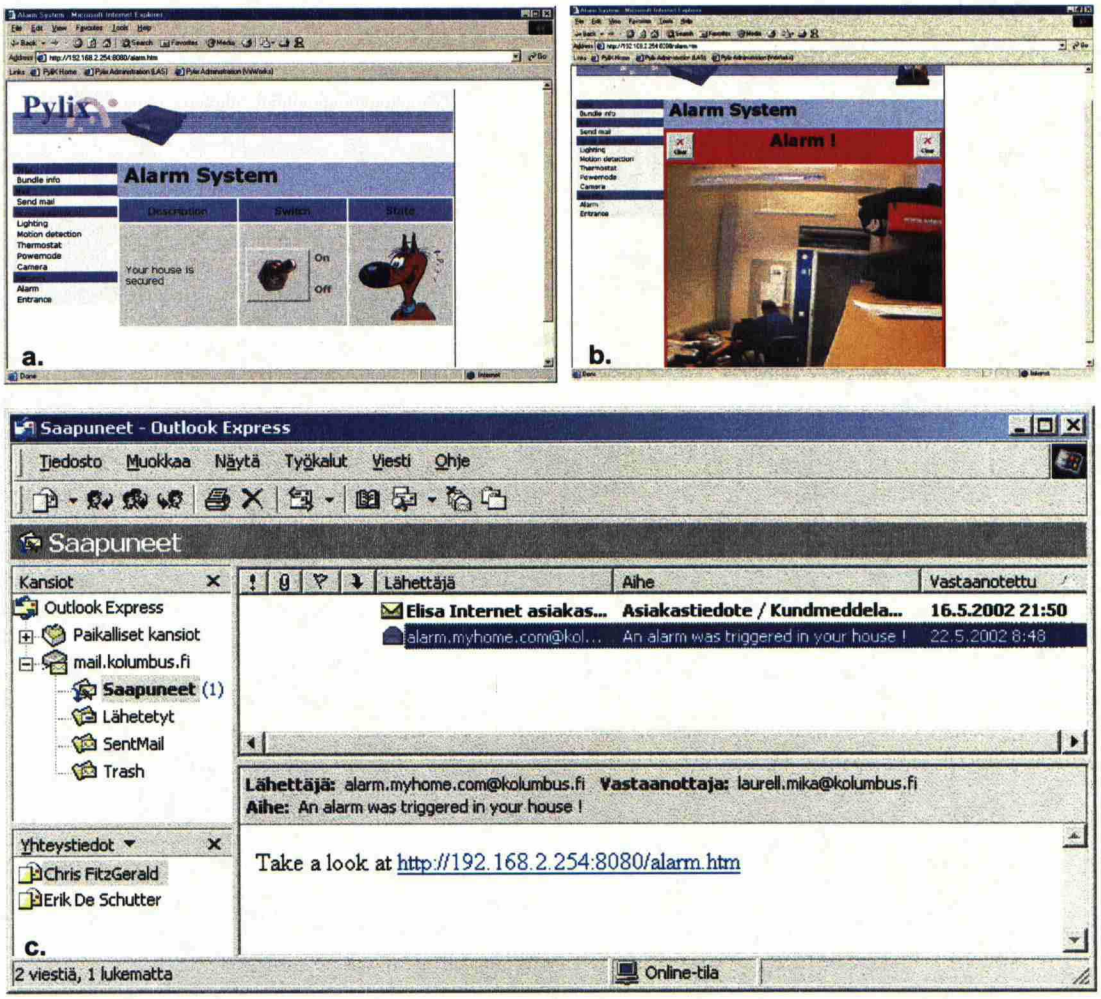
```
# This file stores the properties for the OSGi-applications.

mail.server = mail.kolumbus.fi

axis2100netcam.192.168.2.100 = LivingCam

x10.serialport = COM1
x10.motiondetector.(A,9) = Livingroom Motion
x10.lampmodule.(A,2) = Kitchen Light
x10.lampmodule.(A,3) = Livingroom Light
x10.appliancemodule.(A,4) = Alarm Activator
x10.appliancemodule.(A,5) = Doorbel
x10.appliancemodule.(A,6) = Doorlock
x10.motiondetector.(A,7) = Garage Motion
x10.motiondetector.(A,8) = Kitchen Motion
x10.appliancemodule.(A,1) = Garage Light
```

```
lon.serialport = NOT_COM1
lon.thermostat.01008340C800 = Thermostat
lon.powernode.000580844400 = Powernode

alarm.activator = Alarm Activator
alarm.trigger = Livingroom Motion
alarm.light = Livingroom Light
alarm.mail.from = alarm@myhome.com
alarm.mail.to = matti.meikalainen@kolumbus.fi
alarm.mail.subject = An alarm was triggered in your house !
alarm.mail.text = Take a look at http://192.168.2.254:8080/alarm.htm

telephony.ipaddress = 192.168.2.253
telephony.port = 8555
entrance.phonenumber = 500101
entrance.announcementserver = 500201
```

Tiedostossa määritellään kodin X-10-laitteet, LON-laitteet ja hälytyspalvelun asetukset.

Palvelun toiminta on seuraavanlainen. Palveluportti julkaisee hälytyspalvelun tarjoamat www-sivut. Palvelun sivut käsittävät kaksi erillistä sivua. Ensimmäisellä sivulla hälytyspalvelu voidaan aktivoida ja toisella sivulla käyttäjä voi tarkistaa palvelun tilan. Seuraavaksi esitellään palvelun toiminta käyttökuvauksen avulla.

Käyttäjä on lähdössä aamulla töihin ja avaa hälytyspalvelun etusivun. Hän haluaa aktivoida hälytyksen (Kuva 49a.), koska hän ei ole tulossa kotiin illalla ja hänen kotinsa jää muuten vartioimatta. Päivällä liiketunnistin havaitsee liikettä yhdessä valvottavista huoneista ja ilmoittaa siitä palvelulogiikalle. Palvelu käyttää palvelualustaan liitettyä verkkokameraa ja ottaa valokuvan siitä huoneesta, jossa liikettä havaittiin. Palvelu julkaisee kuvan palvelun tilasivulla (Kuva 49b.) ja lähettää käyttäjälle sähköpostin hälytyksen laukeamisesta.



Kuva 49. Hälytyspalvelun käyttöliittymä.

Käyttäjä vastaanottaa töissä palveluportin lähettämän sähköpostin (Kuva 49c.), jossa hälytyspalvelu ilmoittaa hälytyksen laukeamisesta. Viestin mukana lähetetään linkki, josta käyttäjä näkee verkkokameran ottaman kuvan hälytyksen laukaisun aiheuttaneesta liikkeestä. Käyttäjä avaa linkin ja huomaa, että talonmies on saapunut korjaamaan termostaattia. Ei siis syytä huoleen. Käyttäjä kuittaa hälytyksen ja määrittelee hälytyksen aktivoitumaan uudelleen talonmiehen poistuttua asunnosta.

Koska hälytys voi laueta myös illalla tai yöllä, kun kameralle ei pimeyden takia ole riittävästi valoa kuvien ottamista varten, on palveluun liitetty valojen sytytysautomatiikka. Kun hälytys laukeaa, sytyttää palvelulogiikka sen huoneen valon, jossa liikettä havaittiin. Näin kamera pystyy ottamaan laadukkaita kuvia riippumatta ympäristön valaistusolosuhteista.

Alla oleva lokilistaus kertoo palvelun toiminnan palveluportin kannalta. Kohdassa I hälytyspalvelu aktivoidaan. Palvelu päivittää kaikki siihen liittyvät sivut.

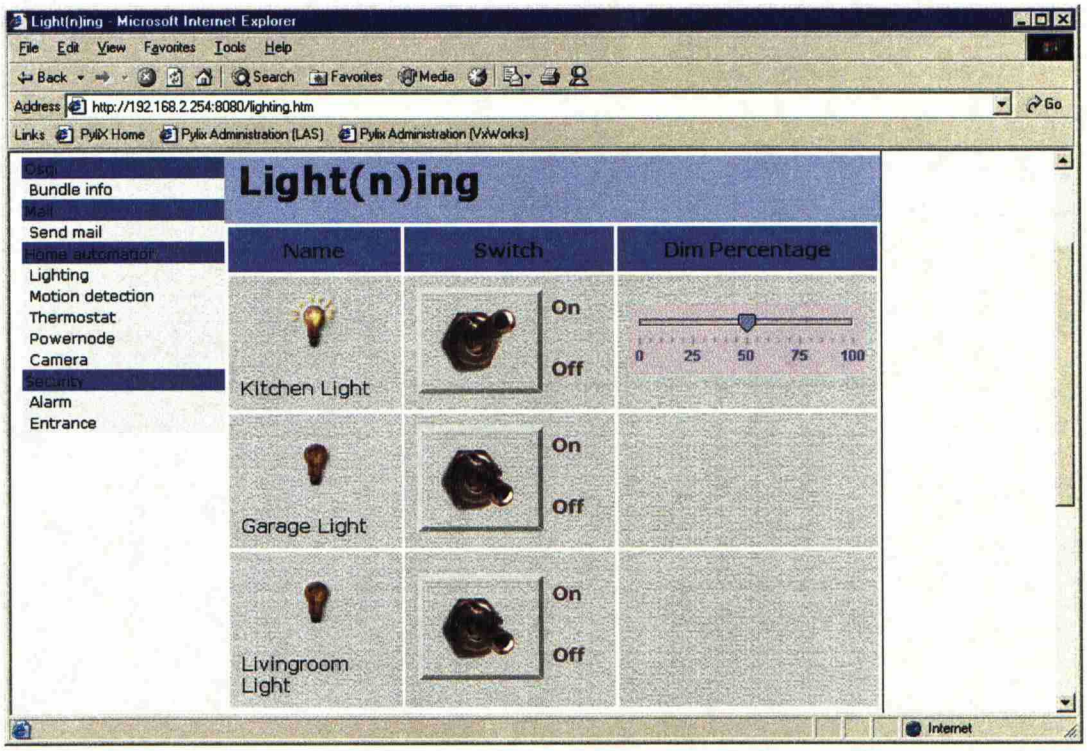
Palvelualusta vastaanottaa ilmoituksen liikkeestä sarjaportin kautta liiketunnistimelta (kohta II). Tämän jälkeen palvelu sytyttää valon ja päivittää valaistuksen hallintasivun (kohta III). Kohdassa IV palvelu ottaa verkkokameralla kuvan liikeilmaisimen laukaisseesta kohteesta ja lähettää sen käyttäjän määrittelemään sähköpostiosoitteeseen. Viimeisenä tehtävänä palvelu julkaisee verkkokameran ottaman kuvan hälytyspalvelun sivulla (Kuva 49b.).

```
    09:46:40,5|FramedServlet   |HttpServer.Http|Starting to get alarm.htm...
    09:46:40,5|FramedServlet   |HttpServer.Http|Getting alarm.htm done.
    09:46:40,7|BrowserConnecti|Thread-14       |Browser closed connection.
    09:46:41,4|RefreshServer   |Thread-2        |A browser connected, waiting for the
              |                |                |alias...Socket[addr=192.168.2.109/
              |                |                |192.168.2.109, port=1247,
              |                |                |localport=55555]
    09:46:41,4|RefreshServer   |Thread-2        |Browser opened page alarm.htm
    09:46:41,4|RefreshServer   |Thread-2        |Registering new browserconnection...
 I  09:46:41,6|RefreshServer   |Thread-2        |Waiting for a browser to connect...
    09:46:49,0|FramedServlet   |HttpServer.Http|Starting to get alarm.htm...
    09:46:49,0|AlarmImpl       |HttpServer.Http|The alarm is armed.
    09:46:49,1|PageRefresher   |HttpServer.Http|Page osgi.htm is being refreshed.
    09:46:49,1|PageRefresher   |HttpServer.Http|Page motion.htm is being refreshed.
    09:46:49,1|PageRefresher   |HttpServer.Http|Page camera.htm is being refreshed.
    09:46:49,1|PageRefresher   |HttpServer.Http|Page alarm.htm is being refreshed.
    09:46:49,1|PageRefresher   |HttpServer.Http|Page entrance.htm is being refreshed.
    09:46:49,1|AlarmFrame      |HttpServer.Http|The alarm system is armed.
    09:46:49,1|FramedServlet   |HttpServer.Http|Getting alarm.htm done.
    09:46:49,3|FramedServlet   |HttpServer.Http|Starting to get alarm.htm...
    09:46:49,4|FramedServlet   |HttpServer.Http|Getting alarm.htm done.
    09:46:49,6|BrowserConnecti|Thread-15       |Browser closed connection.
    09:47:59,9|GatewayImpl     |Thread-8        |state(receiving), received byte(02)
    09:47:59,9|GatewayImpl     |Thread-8        |Received message : 01 62
    09:47:59,9|GatewayImpl     |Thread-8        |Received function-event-byte 02
II  09:47:59,9|MotionDetection|Thread-8        |Now, a refresh of all the browsers
              |                |                |that are viewing this frame should
              |                |                |occur...
    09:47:59,9|PageRefresher   |Thread-8        |Page motion.htm is being refreshed.
    09:47:59,9|AlarmImpl$Trigg|Thread-8        |An alarm is triggered !
    09:47:59,9|AlarmImpl       |Thread-17       |Switching the alarm light.
    09:47:59,9|GatewayImpl     |Thread-17       |Sending x10 command [(A,3)|ON]
    09:47:59,9|GatewayImpl     |Thread-17       |Sending event (04 62), expecting
              |                |                |checksum : 66
    09:47:59,9|GatewayImpl     |Thread-17       |Writing to serial port : 04 62
III 09:47:59,9|GatewayImpl     |Thread-8        |state(expecting checksum), received
              |                |                |byte(66)
    09:48:01,1|LightingFrame$H|Thread-17       |Now, a refresh of all the browsers
              |                |                |that are viewing this frame should
              |                |                |occur...
    09:48:01,1|PageRefresher   |Thread-17       |Page lighting.htm is being refreshed.
    09:48:01,1|AlarmImpl       |Thread-17       |Taking a picture of the scene of the
              |                |                |crime.
    09:48:04,4|AlarmImpl       |Thread-17       |Picture of the crimescene taken.
    09:48:04,4|MailImpl        |Thread-17       |Sending a mail using mailserver
              |                |                |mail.kolumbus.fi
 IV 09:48:04,4|MailImpl        |Thread-17       |  from  alarm@myhome.com
    09:48:04,4|MailImpl        |Thread-17       |  to    matti.meikalainen@kolumbus.fi
    09:48:04,4|MailImpl        |Thread-17       |  subject An alarm was triggered in
              |                |                |          your house !
    09:48:04,4|MailImpl        |Thread-17       |  text  Take a look at
              |                |                |  http://192.168.2.254:8080/alarm.htm
```

```
09:48:13,8|PageRefresher |Thread-17      |Page osgi.htm is being refreshed.
09:48:13,9|PageRefresher |Thread-17      |Page lighting.htm is being refreshed.
09:48:13,9|PageRefresher |Thread-17      |Page motion.htm is being refreshed.
09:48:13,9|PageRefresher |Thread-17      |Page camera.htm is being refreshed.
09:48:13,9|PageRefresher |Thread-17      |Page alarm.htm is being refreshed.
09:48:13,9|PageRefresher |Thread-17      |Page entrance.htm is being refreshed.
09:48:13,9|AlarmFrame    |Thread-17      |Image of the crimescene has been
          |              |               |published.
```

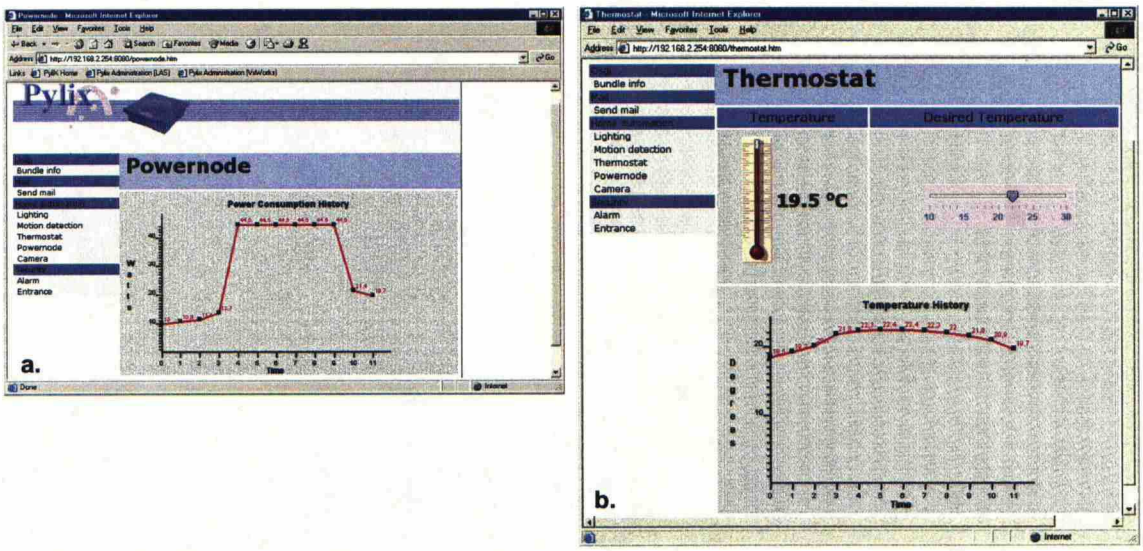
IV

Samalla kun valojen hallinta toteutettiin hälytyspalvelua varten, oli luonnollista lisätä valojen hallinta myös käyttäjän palveluksi. Kuvassa 50 on esitetty valojen hallintakäyttöliittymä, josta käyttäjä pystyy hallitsemaan kaikkia palveluporttiin liitettyjä valaisimia. Käyttäjä voi käyttää valojen hallintapalvelua esimerkiksi kämmenmikron avulla.



Kuva 50. Valaistuksen hallintapalvelun käyttöliittymä.

Palveluporttia voidaan käyttää hyväksi myös monella muulla tapaa. Sähköyhtiö voi tehdä asukkaan kanssa sopimuksen, mikä sallii sähköyhtiön lukea sähkömittarinsa palveluportin avulla. Myös kodin asukas voi hyötyä tästä mahdollisuudesta. Kuvassa 51a. on esitetty, kuinka LON-mittalaittetta käyttäen palveluportti pystyy tuottamaan historiatietoa kodin energian kulutuksesta. Liittämällä tähän palveluun kodin lämpötilan hallinnan (Kuva 51b.) saadaan paljon arvokasta tietoa, kuinka energian kulutusta voidaan säädellä rytmittämällä lämmitys esimerkiksi edulliseen sähkötariffiin.



Kuva 51. Käyttöliittymänäkymät energian kulutus- ja lämpötilanhallintapalveluista.

4.5 Arvio

Vaikka edellä kuvattu hälytyspalvelu on varsin yksinkertainen, eikä ole mitenkään riittävä kodin hälytysjärjestelmäksi, osoittaa se kuitenkin millainen OSGi-palvelualustaan luotavan palvelun rakenne on. Jotta palvelu olisi haluttava, tulee palvelun tarjota jotain enemmän kuin pelkästään valojen vilkuttelua.

Kuitenkin esimerkki osoittaa selkeästi millaisia mahdollisuuksia palvelun tarjoajilla on suunnitella uusia palveluita, jotka käyttävät hyväkseen edullisia palvelualustaan liitettäviä lisälaitteita. Jo tänä päivänä markkinoilla on joukko yrityksiä, jotka tarjoavat edullisia komponentteja kodin automaatiojärjestelmään. Enää ei puutu kuin osaavat palvelun tarjoajat, jotka alkavat toteuttaa näitä komponentteja hyväksikäyttäviä palveluita.

Palvelun toimintakuntoon laittamisen yhteydessä tuli selkeästi esiin palvelualustan tämän hetkinen heikkous. Koska lähes kaikki palvelualustat ovat tällä hetkellä prototyyppejä, niiden luotettavuus oli varsin kyseenalainen. Lisäksi jokainen palveluympäristö vaatii varsin laajan konfiguroinin palvelualustan, palveluiden ja laitteiden osalta, ennen kuin se on valmis käytettäväksi.

Edellä mainittujen seikkojen lisäksi tulee vielä tietoturvakysymykset. Ilman riittävää tietoturvaa ei ole järkevää lähteä tarjoamaan palveluita.

5. TIETOTURVA

Kaikilla järjestelmillä on omat riskinsä. Jotta palveluporttia voitaisiin tarjota kaupallisesti, on sen tietoturvaan liittyvät riskit kartoitettava ja niiden vaikutukset minimoitava. Taulukossa 5 on tunnistettu joitakin palveluporttiin liittyviä tietoturvariskejä. Nämä on jaettu kahteen ryhmään: tietoliikenneturvallisuuteen ja sovellusturvallisuuteen liittyviin riskeihin. Molemmat pääryhmät voidaan jakaa edelleen aliryhmiksi, joita ovat asiakkaan omaan toimintaan, palvelun tarjoajan toimintaan ja ulkopuoliseen toimintaan perustuvat tietoturvariskit.

	Tietoliikenneturvallisuus - *luottamuksellisuus* - *käytettävyys* - *eheys*	**Sovellusturvallisuus** - *kiistämättömyys* - *tarkistettavuus*
Asiakkaan oma toiminta	- salasanojen leviäminen - käyttäjän tekemien asetusten virheet	- laskentatietojen väärennös - palveluiden tahaton tai tahallinen väärinkäyttö - epämääräisten palveluiden lataaminen - virukset ja troijalaiset
Palvelun tarjoajan toiminta	- palomuurien pettäminen - virhe verkkosuunnittelussa - asiakastietojen leviäminen	- virheellinen *bundle* - väärennetty *bundle* - asiakastietojen leviäminen - ylläpitoliittymän hajoaminen - korruptoituneet ylläpitäjät
Ulkoinen toiminta	- kotiverkkoon tunkeutuminen - tekeytyminen toiseksi - palvelunestohyökkäykset	- etähallintarajapinnan murtaminen - *bundle*'ien väärentäminen - palveluihin tai käyttöjärjestelmään murtautuminen - virukset ja troijalaiset

Taulukko 5. Palveluportin tietoturvamatriisi.

Tämän diplomityön tarkoituksena ei ole miettiä ratkaisuja mahdollisiin tietoturvaongelmiin. Tämän vuoksi työssä on keskitytty lähinnä tunnistamaan riskit, jotta niihin voidaan myöhemmin pohtia ratkaisuja.

5.1 Tietoliikenneturvallisuus

Tietoliikenteen turvallisuus koostuu kolmesta perustermistä: luottamuksellisuudesta, käytettävyydestä ja eheydestä. Luottamuksellisuudella tarkoitetaan sitä, ettei tieto ole sellaisten tahojen saatavilla, kenelle se ei ole tarkoitettu. Toisaalta tietojen tulee olla saatavilla eli käytettävissä aina, kun käyttäjä tarvitsee tietoa. Tiedon eheydellä tarkoitetaan tiedon muuttumattomuutta. Kukaan sellainen taho, henkilö tai resurssi, jolla ei ole oikeita oikeuksia, ei pääse muuttamaan tietoja tai tietojen muuttaminen havaitaan esimerkiksi tarkistussummalla.

Palveluportin yhteydessä tämä tarkoittaa taulukossa 5 lueteltujen turvallisuusriskien poistamista. Operaattorin on vaikea puuttua asiakkaan omaan toimintaan perustuviin tietoturvariskeihin, kuten salasanojen leviämiseen ja asetusten virheellisyyteen. Oikeastaan ainoa tapa puuttua näihin on riittävä ja selkeä ohjeistus. Mikä on sitten riittävä ohjeistus, on täysin tapauskohtaista. Tämän takia operaattori voi tarjota asiakkailleen tukipalveluja, jotka avustavat asiakasta epäselvissä tilanteissa.

Palvelun tarjoajan tai operaattorin, joka tarjoaa palveluita palveluporttiin, toiminnasta johtuvat turvallisuusriskit ovat teoreettisesti helpompia ratkaista. Hyvä suunnitteluprosessi, jossa mahdolliset suunnitteluvirheet havaitaan mahdollisimman aikaisessa vaiheessa, auttaa operaattoria vähentämään tuotteeseen päässeitä tietoturva-aukkoja. Käytännössä tämä vaatii suunnittelijoiden ja asentajien perusteellista koulutusta. Silti inhimillisten erehdysten seurauksena tuotteeseen syntyy tietoturva-aukkoja.

Ulkoisesta toiminnasta johtuvat tietoturvariskit ovat tällä hetkellä suurin yksittäinen tietoturvariski. Ulkoiseksi toiminnaksi lasketaan eri tasoisten hakkerien tekemät murtoyritykset, murrot ja erilaiset palveluihin kohdistuvat hyökkäykset. Ulkoisen toiminnan aiheuttamien tietoturvariskien torjunta vaatii varsin paljon resursseja ja tietotaitoa operaattorilta. Esimerkiksi palvelunestohyökkäyksen torjunta vaatii valpasta verkkovalvojaa. Verkkovalvoja voi selvittää hyökkäyksen lähtöpisteen ja estää sieltä saapuvien pyyntöjen käsittelyn operaattorin verkossa.

Ulkoisesta toiminnasta johtuvien riskien arvioinnissa tulee muistaa, että hyökkääjä valitsee monesti kohteensa mielenkiinnon perusteella. Jos suojeltava järjestelmä ei

ole mielenkiintoinen hyökkääjän mielestä, on sitä turha suojata täydellisesti. Ongelmaksi muodostuukin juuri tuon kiinnostavuuden arviointi.

5.2 Sovellusturvallisuus

Sovellusturvallisuus voidaan kiteyttää kahteen pääkohtaan: kiistämättömyyteen ja tarkistettavuuteen eli eheyteen. Kuten tietoliikenteen kohdalla on sovellusturvallisuuttakin helpompi käsitellä, jos se jaetaan kolmeen jo mainittuun alaryhmään.

Asiakas muodostaa edelleen vaikeasti hallittavan turvariskin. Seuraavassa lyhyt esimerkki kuvaa yhtä käyttäjän mahdollisuutta rikkoa sovellusturvallisuutta. Käyttäjä selailee operaattorin palvelulistausta ja siirtyy palveluiden kauppapaikalle. Ostotapahtumassa käyttäjä täyttää HTML-lomakkeella tiedot, minkä palvelun hän on ostamassa. Lomakkeen lopussa on Javascriptillä toteutettu loppusumman laskuri, joka laskee asiakkaalta veloitettavan summan. Järjestelmä voi käyttää SSL-salausta (Secure Sockets Layer) tietojen välittämisessä, jolloin kenenkään ulkopuolisen on käytännössä mahdotonta päästä tietoihin käsiksi.

Järjestelmän tekninen turvallisuustaso on lähetettävien tietojen salaus ja varmistus. Turvallisuus takaa, että protokollat ja ohjelmat toimivat luotettavasti. Tässä kyseisessä esimerkkitapauksessa laskun loppusumman laskenta on toteutettu käyttäjän selaimessa. Nyt onkin mahdollista, että käyttäjä tallentaa HTML-lomakkeen, muokkaa sitä siten, että loppusumma on esimerkiksi aina 10 senttiä. Kun käyttäjä avaa lomakkeen ja tilaa sillä tuotteita, hänen ei tarvitse pelätä suuria laskuja. Ongelma kyseisessä tapauksessa on, ettei ole helppoa tapaa estää näitä turva-aukkoja automaattisesti, koska ne eivät ole teknisiä ja riippuvat käyttäjän toiminnasta.

Koska palvelun tarjoaja vastaa palvelun toiminnasta, on myös palveluporttiin asennettavat *bundle*'it palvelun tarjoajan vastuulla. Ennen kuin palvelun tarjoaja voi tarjota palvelua asiakkailleen, on sen tehtävä palvelulle hyväksyntätarkastus. Tässä tarkastuksessa tulee tulla ilmi palveluun liittyvien *bundle*'ien toiminta. Jos tämä tarkastus suoritetaan vain pintapuolisesti tarkastelemalla, on mahdollista, että vihamielinen taho saa oman esimerkiksi troijalaisena hevosena toimivan *bundle*'in mukaan operaattorin tarjoamiin palveluihin. Troijan hevonen on yksi tapa hyökätä

tietojärjestelmiin. Se ei levitä itse kopioita itsestään, vaan se kykenee leviämään ainoastaan käyttäjän avustuksella. Troijan hevonen voi aiheuttaa vahinkoa niin kuin muutkin virukset esim. tuhoamalla tiedostoja. Eräät troijalaiset ovat keränneet tietoja järjestelmästä ja lähettäneet niitä tekijälleen.

Palvelun tarjoajan toiminnasta johtuvista sovellusturvariskeistä puhuttaessa on syytä muistaa, että huolimatta edistyksellisistä teknologisista ratkaisuista on aina mahdollista, että vihamielinen taho toteuttaa hyökkäyksensä käyttäen hyväkseen taivuttelumenetelmää (social engineering). Tämä voi tapahtua esimerkiksi lahjomalla sopiva henkilö operaattorin verkkovalvonnassa.

Osa ulkopuolisen toiminnan aiheuttamista sovellusturvariskeistä vaatii myös asiakkaan omaa toimintaa. Tällaisia ovat mm. troijalaiset sekä useimmat viruksen tavoin leviävät hyökkäykset. Nämä vaativat aina jonkin asteista käyttäjän toimintaa ennen aktivoitumistaan. Usein kyseessä on varsin harmittomalta näyttävä linkki tai ohjelman nimi, jonka suorittamalla Troijan hevonen tai virus aktivoituu ja mahdollistaa hyökkääjän pääsyn kotiverkkoon.

6. YHTEENVETO JA JOHTOPÄÄTÖKSET

OSGi on teollisuustyöryhmä, jonka tavoitteena on määritellä avoin palveluporttisuositus. Suosituksessa määritellään miten uuden sukupolven älykkäät kotien ja toimistojen koneet ja laitteet voidaan liittää kaupallisten Internet-palveluiden piiriin. OSGi tarjoaa palvelun tuottajille, järjestelmäntoimittajille ja laitevalmistajille avoimen ja yhteisen Java-pohjaisen arkkitehtuurin, jolla he voivat hallitusti kehittää laitteistoriippumattomia sovelluksia ja palveluita. Itse asiassa palveluportti on kokoelma erilaisia ohjelmistorajapintoja, jotka määrittelevät standardin.

Palveluporttimääritys täydentää nykyisiä koti- ja rakennusautomaatioväylästandardeja kuten Jiniä, Bluetoothia, CEBusia tai LonWorksia tarjoamalla niille yhdyskäytävän tietoverkkoihin rakennusten ulkopuolelle. Teknisesti palveluportti on sulautettuun järjestelmään tarkoitettu palvelinohjelmisto, joka sijoitetaan ulkoisen Internet-verkon ja rakennuksen sisäisten verkkojen välille. Palveluportin suunnittelussa on alusta alkaen otettu huomioon se, miten järjestelmä soveltuu kotien ja toimistojen tarvitsemien tiedonsiirtoon pohjautuvien sisältöpalveluiden tuottamiseen. Esimerkkinä sisältöpalveluista voidaan mainita energiayhtiöiden tarjoama energian mittaus ja älykäs kuormituksen ohjaus, tapauskohtaisesti suunnitellut kodin turvajärjestelmän toiminnot tai vaikkapa sairaaloiden ja terveyskeskusten tarjoama vanhusten terveyden tilan jatkuva valvonta mittausjärjestelmineen. [25]

Palveluportin ytimen muodostaa viitekehys (*framework*). Se on Java-ajoympäristön päälle toteutettu palvelualusta. Palvelut toteutetaan viitekehyksessä *bundle*'eilla, jotka ovat ainoita olioita, mihin Java-pohjaisia sovelluksia sijoitetaan. Käytännössä *bundle*'it koostuvat Java-luokista sekä muista resursseista, mitkä yhdessä tarjoavat toimintoja käyttäjille sekä komponentteja eli palveluita muille *bundle*'eille.

Bundle'ien avulla palvelusovellusta voidaan tarvittaessa muuttaa dynaamisesti palvelun ollessa käytössä. Tämä tapahtuu ilman viitekehyksen uudelleen käynnistämistä. Tämän lisäksi *bundle*'ien käyttö mahdollistaa palvelun lataamisen

viitekehykseen vasta, kun palvelua tarvitaan ja sen poistamisen palvelun käytön jälkeen. Näin voidaan tarjota monipuolisia palvelukokonaisuuksia jopa pienellä muistilla varustettuihin laitteisiin.

Palveluporttimäärityksessä OSGi on valmiiksi määritellyt joukon rajapintoja, joita palvelun suunnittelijat voivat käyttää omassa palvelussaan. Viitekehyksen rajapintojen standardoinnin ajatuksena on, ettei jokainen toteuta rajapintaa omalla tavallaan ja näin lisää palveluiden toimittajariippuvuutta, vaan rajapinnat ovat yhteiset ja jokaisen vapaassa käytössä. Vastaavanlainen rakenne on myös suositeltava palvelun tarjoajan tuottamissa palveluissa.

Palveluiden tarjoaminen voidaan jakaa rooleihin. Näistä rooleista keskeisimpiä ovat operaattorin, palvelun tarjoajan, palvelun kokoajan ja palvelun yhdistäjän roolit. Operaattorin vastuulla on koko toimintaketjun toiminta. Palvelun tarjoajat ovat erilaisia ohjelmistoyrityksiä tai sisällöntuottajia, mitkä tarjoavat palveluita käytettäväksi palveluporteissa. Palvelun kokoajan ja yhdistäjän roolit ovat hyvin läheiset. Palvelun kokoaja kerää palvelumäärityksen toteuttavat komponentit. Palvelun yhdistäjä tekee komponenteista yhtenäisen palvelukokonaisuuden. Esimerkkinä palvelusta on uutispalvelu, missä uutiset kerätään eri tietotoimistoilta ja niihin lisätään vielä sääennusteita tuottavan yrityksen sääennuste.

Jotta palveluporttimalli olisi asiakkaasta houkutteleva, tulee edellä mainittujen tahojen toimia saumattomasti yhteen. Tästä vaatimuksesta johtuen operaattorilla on suuri vastuu valitessaan yhteistyökumppaneita palvelun toimitusketjuun.

Palveluporttiin liittyvien palveluiden käytöstä laskuttaminen on vielä hieman avoinna. Erilaisia laskentaperusteita, kuten kiinteää laskutusta, aktivoinnista tai sisällöstä tuotettavaa laskutustietoa tai edellisten yhdistelmiä, on esitelty liitettäväksi OSGi-palvelualustamääritelmään. Valinta, miten palvelusta tullaan laskuttamaan, on palveluoperaattorin harkinnassa.

Palveluporttiin ollaan määrittelemässä laskentatietojen keräyspalvelua. Sen tarkoitus on tuottaa laskentaoperaattorille eli operaattorille, joka tuottaa asiakkaalle laskun, sen laskentajärjestelmän tarvitsemia tietoja palveluiden käytöstä, asentamisesta sekä poistamisesta.

Jotta palveluportteja voidaan liittää joustavasti operaattorin verkkoon, on niiden tuettava etähallintaa. OSGi-määrityksen versioon 2.0 perustuvissa palveluporteissa etähallinta on vielä toteutettu valmistajakohtaisilla ratkaisuilla, mutta tulevaisuudessa etähallinta on osa OSGi:n virallista avoimen palvelualustan määritystä.

Jotta palvelun tuottamisen ja sen toimittamisen muodostama kokonaisuus havainnollistuisi on työssä toteutettu esimerkkipalvelu. Palvelun muodostaa palveluporttiin liitetty kodinautomaatiojärjestelmä, verkkokamera ja muutama tarkoitusta varten kirjoitettu *bundle*. Tavoitteena oli luoda yksinkertainen kodin vartiointipalvelu, missä liiketunnistimen havaitessa liikettä kotona suoritetaan siitä hälytys käyttäjän haluamaan paikkaan. Hälytysviestin mukana välitetään käyttäjälle verkkokameran ottama kuva hälytyksen aiheuttajasta. Palvelun sivutuotteena syntyi myös kotiautomaatioon liittyvä valaistuksen säätöpalvelu.

Tietoturva on merkittävässä roolissa mietittäessä palveluporttien laajempaa käyttöä ja niihin liittyviä palveluita. Tässä työssä ei ole mietitty tietoturvaongelmille ratkaisuja, vaan on keskitytty lähinnä tunnistamaan ongelmat. Tunnistetut ongelmat jaettiin kahteen ryhmään: tietoliikenneturvallisuuteen ja sovellusturvallisuuteen. Jos palveluportteja aletaan yksittäisiä kokeiluja laajemmin ottamaan käyttöön, tulee näihin tässä työssä tunnistettuihin ongelmiin löytää ratkaisut.

OSGi on tehnyt paljon hyvää työtä tehdessään palvelualustamäärityksen toisen version. Heidän oman ennustuksensa mukaan seuraava versio palvelualustamäärityksestä julkaistaan vuoden 2002 aikana. Siihen on tarkoitus lisätä muutamia varsin oleellisia palveluita, joita aikaisemmissa määrityksissä ei ole ollut, kuten tuki julkisen avaimen käytölle (Public Key Infrastructure, PKI) ja käyttöönottoasennus (Initial setup). Molemmat mainituista lisäyksistä ovat varsin olennaisia palveluita, kun palveluportteja toimittetaan käyttäjille. Näiden puuttuvien palveluiden määrittäminen ei kuitenkaan merkittävästi vaikuta nykytilanteeseen, sillä jo nyt kaupalliset OSGi-viitekehykset, kuten Prosyst'in mBedded Server ja Gatespace'n e-Service, ovat toteuttaneet edellä mainitut palvelut. Toteutukset ovat tosin valmistajakohtaiset, joten yhteensopivuudesta ei ole varmuutta.

Palveluporttikonseptiin jäi vielä joitakin kysymyksiä. Suurimpana mieleen nousee OSGi:n tarvitsema laite. Kannattaako asiakkaille tarjota erillistä laitetta, joka tarjoaa ainoastaan mahdollisuuden uusien palveluiden käyttöön? Tämä ei todennäköisesti ole kannattavaa, vaan viitekehys, ja miksi ei koko palveluportti, kannattaa yhdistää jonkin toisen kodin päätelaitteen kanssa yhteen laatikkoon. Sopiva laite, jonka kanssa palvelualusta voitaisiin yhdistää voisi olla digitaalisen television tv-sovitin tai uuden puhelinverkkotekniikan (Next Generation Networks, NGN) tarvitsema verkkosovitin. Molemmat mainituista laitteista tuovat mukanaan jonkin asiakkaan haluaman palvelun. Tällöin OSGi voi olla eräänlainen kaupanpäällinen, jonka avulla operaattori voi toteuttaa asiakkaalle uusia houkuttelevia palveluita.

Muutenkin ajatus palveluportista asettaa operaattorille paljon haasteita. Operaattorin tulee varmistua siitä, että kaikki sen tarjoamat palvelut toimivat kaikissa sen asiakkaiden hallussa olevissa palveluporteissa. Lisäksi ehkä vielä tärkeämpänä seikkana operaattorin tulee varmistua palvelun turvallisuudesta. Tämä tarkoittaa sitä, että operaattorilla tulee olla riittävä osaaminen palveluiden ohjelmakoodin verifioimiseen, ennen kuin palvelu voidaan ottaa jakeluun.

Jo tällä hetkellä operaattoreille on tarjolla ohjelmistoja koodin verifioimiseen. Tulevaisuudessa nämä ohjelmistot varmasti helpottavat merkittävästi operaattorin työtä palvelun tarkastusvaiheessa. Nykyisin ne ovat lähinnä hyvä apu tarkasteltaessa palvelun toimivuutta.

Kokonaisuutena palvelualustaympäristö on vielä varsin keskeneräinen. Siitä puuttuu toimiva ja standardoitu laskentaympäristö, etähallinta, julkisen ja yksityisen avainjärjestelmän tuki sekä monia muita operaattorille tärkeitä ominaisuuksia. Osa edellä mainituista puutteista korjaantuu seuraavassa palvelualustaversiossa, mutta esimerkiksi laskentapalvelun standardointi on vielä pahasti kesken. Tällä hetkellä viitekehystoimittajat ovat toteuttaneet laskennan toimittajakohtaisilla ratkaisuilla. Tämä johtaa siihen, että operaattorin olisi tehtävä varsin merkittäviä muutoksia omissa laskentajärjestelmissään, ennen kuin ne olisivat yhteensopivat toimittajien laskentamallien kanssa.

Toinen vakava puute on laitteiden asennusvaiheen automatisointi. Ei ole vielä olemassa mitään määritelmää, kuinka laite saataisiin automaattisesti asettamaan

oikeat asetukset riippuen käyttäjän valinnoista operaattorin tai palvelun toimittajan kanssa. Tämä tarkoittaa käytännössä sitä, että jos operaattori päättäisi ruveta tarjoamaan palveluporttipalveluita asiakkailleen, tarvitsee operaattorin hallita jokaisen laitteen asennus erikseen.

Vaikka tällä hetkellä OSGi:n palvelualustakonseptilla on lukuisia puutteita, on todennäköistä, että se tulee tarjoamaan yhdessä esim. digi-tv –sovittimen kanssa asiakkaalle monia mahdollisuuksia nauttia uusista elämää helpottavista ominaisuuksista kotona.

7. VIITTEET

[1] Gupta S., Home Networking, White Paper, Wipro Technologies, http://www.wipro.com/shortcuts/downloads.htm, 15.3.2002.

[2] Acunia, The Open Telematics Framework - markkinointi materiaali, Acunia NV, 2002

[3] About OSGi, http://www.osgi.org/about/, 10.10.2001

[4] O'Driscoll G., The Essential Guide to Home Networking Technologies, Prentice-Hall Inc., 2001, 339 s.

[5] OSGi, The Managed Service Specification markkinointi materiaali, The Open Service Gateway Initiative, 2001, 6 s.

[6] Peltonen H., Olio-ohjelmoinnin perusteet C++, Suomen ATK.kustannus Oy, 1996, 389 s.

[7] Wikla A., Ohjelmoinnin perusteet Java-kielellä, OtaDATA ry, 1998, 270 s.

[8] Lewis J., Loftus W., Java software solutions, Addison Wesley, 1998, 857 s.

[9] Sun Microsystems, Managing Source and Class Files, Sun Microsystems, http://java.sun.com/docs/books/tutorial/java/interpack/managingfiles.html, 8.5.2002

[10] OSGi, OSGi Service Platform, The Open Service Gateway Initiative, 2001, 282 s.

[11] Sun Microsystems, JAR File Specification, Sun Microsystems, http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html#JAR%20Manifest, 15.12.2001

[12] IETF, RFC 1960: A String Representation of LDAP Search Filters, IETF, http://www.ietf.org/rfc/rfc1960.txt, 22.2.2002

[13] Luehe J., Reed B., OSGi RFC 12 – PackageAdmin Service for the Core Framework, The Open Service Gateway Initiative, 2001, 10 s.

[14] Reed B., Kriens P., Hargrave B. J., OSGi RFC 11 - ServiceTracker, The Open Service Gateway Initiative, 2001, 16 s.

[15] Bohlin T., OSGi RFC 8 – Device Access, The Open Service Gateway Initiative, 2001, 34 s.

[16] Reed B., Sparud J., Luehe J.,OSGi RFC 10 – User Management API, The Open Service Gateway Initiative, 2001, 25 s.

[17] Microsoft, Plug and Play External COM Device Specification, Microsoft, http://www.microsoft.com/hwdev/download/respec/pnpcom.rtf, 10.7.2001

[18] Global Inventures, Telecommunications Operators Deployment of OSGi Enabled Services, The Open Service Gateway Initiative, 2002, 51 s.

[19] Helander L.-E., OSGi RFC 28 – OSGi Entities, The Open Service Gateway Initiative, 2001, 18 s.

[20] Turtiainen E., Sähköinen kaupankäynti, Teknillinen Korkeakoulu, 1995, 7 s.

[21] Eriksson L., Bohlin T., OSGi RFC 14 - Accounting Service, The Open Service Gateway Initiative, 2002, 45 s.

[22] OSGi RFC 20 - Remote management architecture, The Open Service Gateway Initiative, 2000, 11 s.

[23] Mikä on X-10?, IT-Keskus Oy, http://www.it-keskus.fi/, 3.6.2002

[24] Persson H., Beland B., OSGi RFC 18 – Security Architecture Specification, The Open Service Gateway Initiative, 2001, 17 s.

[25] Pakanen J., Uudet tietoliikenne ratkaisut – Internet kiihdyttää kotiautomaatiota, Prosessori 6-7 1999, s.54-58

8. KUVAT JA TAULUKOT

LIITE A

OSGi ryhmän jäsenet ja perustaja jäsenet.

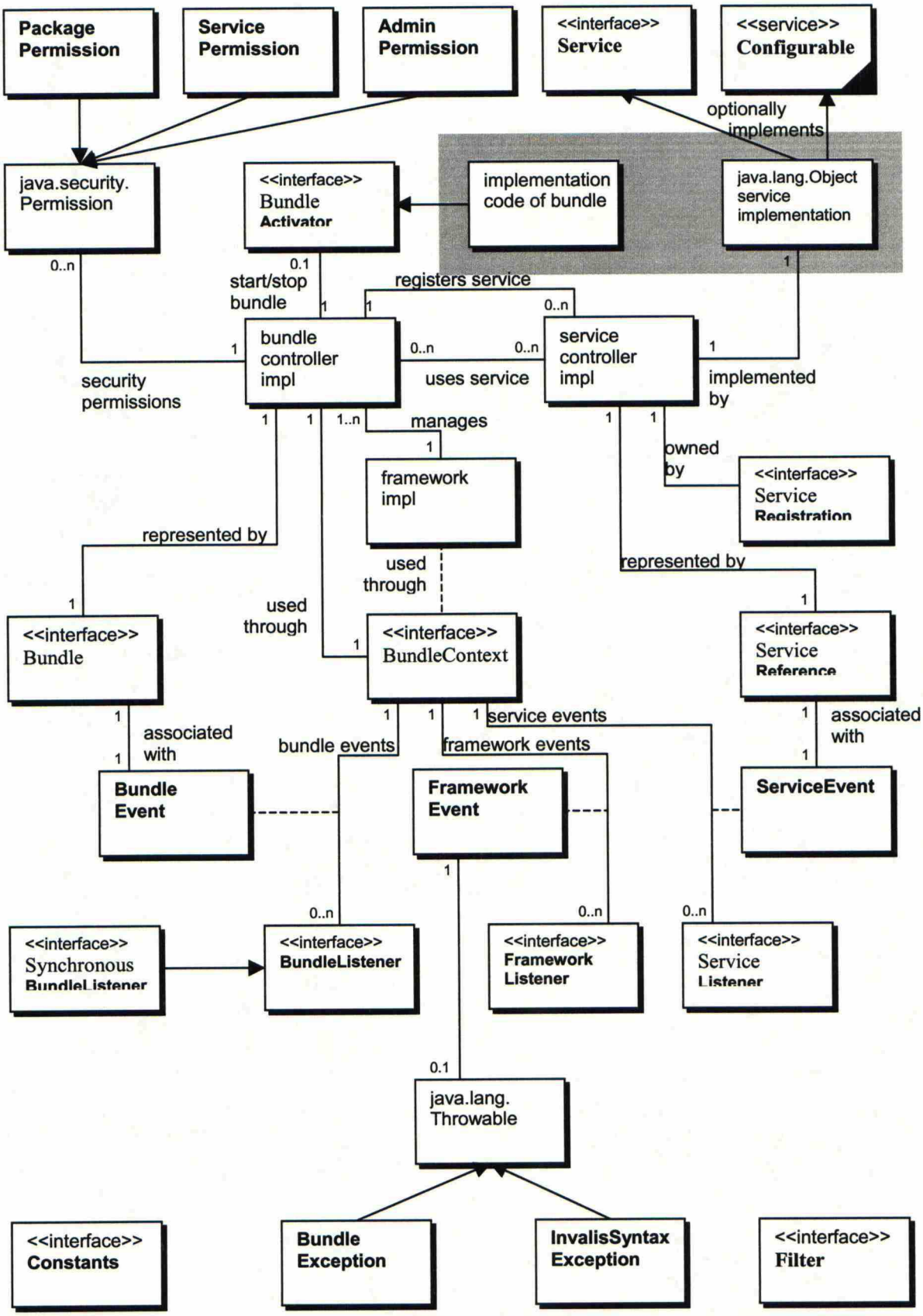
2Wire Inc.
4DHomeNet, Inc.
ABB Corporate Research Ltd.
Acronet Corp
Acunia
AMI-C
BellSouth Telecommunications, Inc.
BMW
Bombardier Transportation
Cablevision Systems
Coactive Networks
Deutsche Telekom
e2-Home AB
Echelon Corporation
Electricite de France (EDF)*
Elisa Communications Corporation
emWare
Ericsson*
Espial Group, Inc.
ETRI
France Telecom
Gatespace AB
Hewlett-Packard
Home Director, Inc.
IBM Corporation*
Infomatec AG
inSilicon Corporation (Phoenix Technologies, ltd.)
Invensys Controls
ITP AS
Jentro AG
KDD R&D Laboratories Inc.
LANergy, Ltd.
Legend Computer System Ltd.
Lineo, Inc.
Lucent Technologies*
Mitsubishi Electric Corporation
Metavector Technologies
Motorola, Inc.*
Netpliance
Nokia Corporation
Nortel Networks*
Novell, Inc.
Oracle Corporation*
P&S Datacom Corporation
Panasonic
Patriot Scientific Corporation
Philips*
ProSyst Software AG
Qwest
Samsung Electronics Co., LTD
SBC Technology Resources, Inc.
Schneider Electric SA
Sharp Corporation
Shell International - ETAC
Sonera Corporation
Sony Corporation
Sprint Communications Company, L.P.
Sun Microsystems*
TAC AB
Telcordia Technologies
Telefonica I+D
Telia
Texas Instruments, Inc.
Tokyo Electric Power Company
Toshiba Corporation*
Tridium, Inc
Ucentric Systems, Inc
Union Fenosa
VDO Car Communications
Verizon
Villa Montage Systems
Whirlpool Corporation
Wind River Systems

Tähdellä merkittyjen perustaja jäsenten lisäsi seuraavat yritykset olivat perustamassa ryhmää:

- Alcatel
- Cable & Wireless
- Enron Communications
- Liberate Technologies
- Sybase

LIITE B

OSGi-viitekehyksen luokkakaavio.



Ohjelmakoodilistaus alarm-*bundle*'ista.

```
package com.metavectortech.alarm.impl;

import java.util.*;
import com.metavectortech.mail.api.*;
import com.metavectortech.alarm.api.*;
import org.osgi.framework.*;

public class AlarmImpl implements FinderListener, AlarmService,
Runnable {

    public AlarmImpl( BundleContext pContext ) {
     mContext = pContext;

     PropertiesFile lPropertiesFile = PropertiesFile.getInstance(
"osgi.demo.properties" );
 ...

     mMailFrom    = lPropertiesFile.getProperty( "alarm.mail.from" );
     if ( mMailFrom == null ) mMailFrom = "alarm@mypylix.com";
     mMailTo      = lPropertiesFile.getProperty( "alarm.mail.to" );
 ...

    }

    public void arm() {
     if ( ! mIsArmed ) {
 T.race( this, T.INFO, "The alarm is armed." );
 mIsArmed = true;
 this.notifyListeners( new AlarmEvent( AlarmEvent.ARM ) );
 ...

    /**
    * This method is called when motion is detected.
    */
    public void run() {
     if ( mIsArmed && ( ! mIsTriggered ) ) {

         mIsTriggered = true;

         // Switching the light
         try {
      if ( mLight == null ) {
         T.race( this, T.INFO, "No alarm-light available." );
      } else {
              T.race( this, T.INFO, "Switching the alarm light." );
              mLight.turnOn();
              ...

         // Taking the picture of the camera
         try {
      if ( mCamera == null ) {
        T.race( this, T.INFO, "No alarm-camera available." );
          mCrimeSceneJpgData = null;
         } else {
   // Create a picture of the scene of the crime
              T.race( this, T.INFO, "Taking a picture of the scene of
the crime." );
             mCrimeSceneJpgData = mCamera.getImageJpgData( "640x480"
);
```

```
                    T.race( this, T.INFO, "Picture of the crimescene
taken." );
        ...

         // Send the mail
           try {
       if ( ( mMail == null )
          || ( mMailTo == null ) ){
          T.race( this, T.INFO, "No alarm-mailservice available." );
          } else {
               mMail.send( mMailFrom, mMailTo, mMailSubject, mMailText
);
        ...
```